Document Identifier: DSP0267

Date: 2023-12-13

Version: 1.3.0

Platform Level Data Model (PLDM) for Firmware Update Specification

Supersedes: 1.2.0

Document Class: Normative

Document Status: Published

Document Language: en-US

CONTENTS

Figures

Tables

Foreword

The *Platform Level Data Model (PLDM) for Firmware Update Specification* (DSP0267) was prepared by the Platform Management Communications Infrastructure (PMCI) Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about DMTF, see https://www.dmtf.org/.

Acknowledgments

DMTF acknowledges the following individuals for their contributions to this document:

Editor:

- Patrick Caporale – Lenovo

Contributors:

- Richelle Ahlvers – Broadcom Inc.
- Scott Dunham – Lenovo
- Kaijie Guo – Lenovo
- Brett Henning – Broadcom Inc.
- Yuval Itkin – NVIDIA Corporation
- Ira Kalman – Intel Corporation
- Shai Lazmi – QLogic Corporation
- Eliel Louzoun – Intel Corporation
- Rob Mapes – Marvell International Ltd
- Balaji Natrajan – Microchip Technology Inc.
- Edward Newman – Hewlett Packard Enterprise
- Jeffrey Plank – Microchip Technology Inc.
- Patrick Schoeller – Hewlett Packard Enterprise, Intel Corporation
- Hemal Shah – Broadcom Inc.
- Tom Slaight – Intel Corporation
- James Smart – Broadcom Inc.
- Bob Stevens – Dell Technologies
- Supreeth Venkatesh – ARM Inc.

Introduction

The Platform Level Data Model (PLDM) Firmware Update Specification defines messages and data structures for updating firmware or other code objects maintained within the firmware devices of a platform management subsystem. Additional functions related to the sequence of identifying and transferring the firmware, are also defined.

Typographical conventions

The following typographical conventions are used in this document:

- Document titles are marked in *italics*.

1 Scope

This specification defines messages and data structures for updating firmware or other objects maintained within, or downstream of, a firmware device of a platform management subsystem. Additional functions related to the sequence of identifying and transferring the component image, are also defined. This document does not specify the operation of PLDM which is described in DSP0240.

This specification defines the requirements to access and use PLDM for Firmware Update in a system that supports firmware updates using PLDM. This specification does not specify whether a given system is required to implement that capability. However, if a system does support firmware updates over PLDM or other functions described in this specification, the specification defines the requirements to access and use those functions over PLDM. The implementation and capability discovery of the PLDM for firmware update in the system is outside the scope of this specification. Portions of this specification rely on information and definitions from other specifications, which are identified in Clause 2.Two of these references are particularly relevant:

- DMTF DSP0240, *Platform Level Data Model (PLDM) Base Specification*, provides definitions of common terminology, conventions, and notations used across the different PLDM specifications as well as the general operation of the PLDM protocol and message format.
- DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes Specification*, defines the values that are used to represent different type codes defined for PLDM messages.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated or versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

ANSI/IEEE Standard 754-1985, *Standard for Binary Floating-Point Arithmetic*

DMTF DSP0236, *MCTP Base Specification 1.3*, https://dmtf.org/sites/default/files/standards/documents/DSP0236_1.3.pdf

DMTF DSP0240, *Platform Level Data Model (PLDM) Base Specification 1.1*, https://dmtf.org/sites/default/files/standards/documents/DSP0240_1.1.pdf

DMTF DSP0241, *Platform Level Data Model (PLDM) Over MCTP Binding Specification 1.0*, https://dmtf.org/sites/default/files/standards/documents/DSP0241_1.0.pdf

DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes Specification 1.2*, https://dmtf.org/sites/default/files/standards/documents/DSP0245_1.2.pdf

DMTF DSP0248, *Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification 1.2*, https://dmtf.org/sites/default/files/standards/documents/DSP0248_1.2.pdf

DMTF DSP0249, *Platform Level Data Model (PLDM) State Set Specification 1.1*, https://dmtf.org/sites/default/files/standards/documents/DSP0249_1.1.pdf

IETF RFC2781, *UTF-16, an encoding of ISO 10646*, February 2000, https://www.ietf.org/rfc/rfc2781.txt

IETF RFC4122, *A Universally Unique IDentifier (UUID) URN Namespace*, July 2005, https://www.ietf.org/rfc/rfc4122.txt

IETF RFC4646, *Tags for Identifying Languages*, September 2006, https://www.ietf.org/rfc/rfc4646.txt

IETF STD63, *UTF-8, a transformation format of ISO 10646*, November 2003, https://www.ietf.org/rfc/std/std63.txt

ISO 8859-1, *Final Text of DIS 8859-1, 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No.1,* February 1998

ISO/IEC Directives, Part 2, *Principles and rules for the structure and drafting of ISO and IEC documents*, https://www.iso.org/sites/directives/current/part2/index.xhtml

3 Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 7. The terms in parentheses are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Clause 7 specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 6.

The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

Refer to DSP0240 for terms and definitions that are used across the PLDM specifications. For the purposes of this document, the following additional terms and definitions apply.

3.1
activation

A process in which the firmware device prepares the newly transferred component images to become the active running firmware components.

3.2
auto-apply

A firmware device procedure which is implemented if the component image was being directly placed into the final memory destination in parallel while the component image was being transferred.

3.3
automatic activation

A process whereby the firmware device automatically activates a transferred component image during the apply stage of the firmware update process.

3.4
AC power cycle

A process whereby a complete removal of power to the firmware device is performed.

A common example is a power supply AC cord removed from the system. This will cause all power inputs to the firmware device (including any auxiliary voltage inputs) to be removed.

3.5
AC power cycle activation

A process whereby a firmware device activates any pending firmware component images which indicated an AC power cycle as its activation method.

3.6
code image

A collection of bytes typically executed on a processor to perform a function, which may also include non-executable data.

3.7
component classification

The general type of component.

Values for this field are aligned with the Value Map from CIM_SoftwareIdentity.Classifications. Refer to Table 32 for values

3.8
component comparison stamp

A value that can be used to determine if a given component is a higher or lower version than another value using an unsigned integer comparison.

3.9
component identifier

A vendor defined value which distinguishes between firmware components which may have identical classifications but require different component images.

3.10
component image

A code image contained in a PLDM firmware update package associated with a firmware component of a firmware device.

The component image is transferred to the firmware device using PLDM commands and placed (perhaps in a modified form) into local storage used by the firmware component.

3.11
component image set

One or more component images contained in a firmware update package that are associated with a particular firmware device.

3.12
device identifier record

A set of descriptors used to identify a type of firmware device.

3.13
downstream device

A device that does not directly communicate with an update agent, but can be used in conjunction with a firmware device proxy to enable inventory and update of its firmware component.

3.14
DC power cycle

A process whereby the firmware device has its non-auxiliary power input removed.

As most PLDM termini are contained within a device such as an ASIC or FPGA, those devices may contain an auxiliary and non-auxiliary power inputs. Auxiliary voltage inputs are typically not affected by a DC power cycle and may continue to be energized during the activation process.

3.15
DC power cycle activation

A process whereby the firmware device activates any pending firmware component images which indicated a DC power cycle as its activation method.

3.16
firmware

One or more code images stored within a local memory structure (such as a Flash NVRAM) and accessible by a firmware device.

3.17
firmware device
FD

A PLDM endpoint (terminus) which contains one or more processor elements which execute firmware.

The firmware device interacts with the update agent to perform firmware updates of its resident firmware components. Typically this may be a PCI I/O device.

3.18
firmware device proxy
FDP

A PLDM endpoint (terminus) which is a firmware device that supports one or more downstream devices. The firmware device proxy interacts with the update agent to perform an update of the firmware component contained within any of its attached downstream devices. The firmware device proxy processes PLDM commands/responses/events for firmware update on behalf of the downstream devices.

3.19
firmware component

A logical entity representing a functional portion of a firmware device.

A firmware device may contain one or more firmware components each of which contains a code image that is represented by a component classification, component identifier, and version information. A firmware component may contain both an active and pending code image.

3.20
firmware package header

A collection of fields which describe the contents of a firmware update package and for which firmware devices the firmware update package is applicable.

3.21
firmware update baseline transfer size

The minimum amount of data that can be requested by a firmware device in an individual command when transferring a component image.

3.22
firmware update package

A firmware package header describing the contents concatenated with one or more component images for one or more firmware devices and/or downstream devices.

3.23
individual firmware device package

A standalone package which contains a collection of fields to describe a single or common set of devices concatenated with a single component image.

The individual firmware device package could be used by a firmware update package creator to assemble multiple firmware devices, downstream devices and individual firmware device descriptors and components into a single firmware update package that can be sent to a UA for processing.

3.24
medium-specific reset

A process whereby a firmware device is reset via the specific type of interface that the PLDM terminus within the firmware device uses to communicate.

For example, a PCI device would have a medium-specific reset via a PCI-reset signal. The firmware device will activate any pending firmware component images which indicated a medium-specific reset as its activation method.

3.25
pending firmware component

A new component image has been transferred to the firmware device and it has completely exited the update process (the firmware device is back to IDLE state) but the activation of the component image requires further action to enable the pending images to become the actively running code images.

The firmware component will report details on the pending image (such as version, date, and its activation methods). The applicable activation method shall be performed for the pending image to become the actively running image.

3.26
self-contained activation

Capability of a firmware device whereby the newly transferred component images can immediately become the actively running firmware component code image after receiving an activate command from the update agent.

In some cases a firmware component is not actively running (i.e., a UEFI driver which only executes on system startup) and therefore the self-contained activation will still apply.

3.27
software bundle

One of the component classification values which represents a single component image containing multiple code objects each of which would be known only be the firmware device.

The layout of the code objects within the software bundle is not defined in this spec.

3.28
system reboot

A process whereby the firmware device, which may typically be contained within a platform that has a host operating system, is restarted.

The firmware device will activate any pending firmware component images which indicated a system reboot as its activation method.

3.29
update agent
UA

A PLDM endpoint (terminus) which orchestrates passing component images from a firmware update package to a firmware device.

Typically this agent is contained within a management controller.

4 Symbols and abbreviated terms

The abbreviations defined in DSP0004, DSP0223, and DSP1001 apply to this document. Refer to DSP0240 for symbols and abbreviated terms that are used across the PLDM specifications. The following additional abbreviations are used in this document.

4.1
FD
Firmware Device

4.2
FDP
Firmware Device Proxy

4.3
UA
Update Agent

5 Conventions

Refer to DSP0240 for conventions, notations, and data types that are used across the PLDM specifications.

5.1 Reserved and Unassigned Values

Unless otherwise specified, any reserved, unspecified, or unassigned values in enumerations or other numeric ranges are reserved for future definition by the DMTF.

Unless otherwise specified, numeric or bit fields that are designated as reserved shall be written as 0 (zero) and ignored when read.

5.2 Byte Ordering

Unless otherwise specified, as for all PLDM specifications byte ordering of multi-byte numeric fields or multi-byte bit fields is "Little Endian" (that is, the lowest byte offset holds the least significant byte, and higher offsets hold the more significant bytes).

6 PLDM for Firmware Update Version

The version of this Platform Level Data Model (PLDM) for Firmware Update shall be 1.3.0 (major version number 1, minor version number 3, update version number 0, and no alpha version).

In response to the GetPLDMVersion command described in DSP0240, the reported version for Type 5 (PLDM for Firmware Update, this specification) shall be encoded as 0xF1F3F000.

7 PLDM for Firmware Update Overview

This specification describes the operation and format of request messages (also referred to as commands) and response messages for updating firmware components of a firmware device (FD)

contained within a platform management subsystem. In addition, certain devices that are downstream of an FD can also be updated with this specification as the FD can act as a proxy on the downstream device behalf. These messages are designed to be delivered using PLDM. This specification also permits a subset of commands to be implemented by a firmware device which only supports the reporting of existing firmware component details, without the ability to perform a firmware update. Traditionally, device firmware has been updated by a combination of update tools and binary files provided by individual device manufacturers. Those update tools normally operate inside a host operating system (e.g. Linux/Windows/DOS), whereby each device may have their own method provided by the device manufacturers to update the firmware into flash chips on the device board. This specification identifies a common method to use PLDM for transferring, and activating one or more component images to an FD or downstream device within the PLDM subsystem and thereby avoiding the use of host operating system based tools and utilities.

The basic format that is used for sending PLDM messages is defined in DSP0240. The format that is used for carrying PLDM messages over a particular transport or medium is given in companion documents to the base specification. For example, DSP0241 defines how PLDM messages are formatted and sent using MCTP as the transport. The Platform Level Data Model (PLDM) for Firmware Update Specification defines messages that support the following items and capabilities:

- Component Image Transfer
  - Component image transfer mechanism does not require FD or downstream device specific logic in the UA
  - For an individual firmware device, a firmware update package may contain
    - A single combined component image (component classification of Software Bundle)
    - A single component image for a single firmware component
    - Multiple component images for multiple firmware components that are applicable to the same firmware device
  - For an individual downstream device supported by a FDP, a firmware update package may contain
    - A single combined component image
  - For multiple downstream devices supported by a FDP which support the same component image, a firmware update package may contain
    - A single component image which the FDP can transfer to all applicable downstream devices without the need for the UA to provide the component image multiple times
  - Transfer of a component image is requested through an offset-based method as directed by the FD

- Firmware Update Package to Firmware Device association
  - A mechanism to determine which type of FD a firmware update package is targeted
  - A mechanism to distinguish between firmware update packages applicable to different instantiations of the same FD (e.g. planar vs. adapter)
  - A mechanism to identify the component image that is to be transferred based on device identifier records. A device identifier record may be based on PCI IDs, IANA ID, UUID, or a vendor specific ID.

- Firmware Update Package to Downstream Device association
  - A mechanism to determine which type of downstream device a firmware update package is targeted
  - A mechanism to distinguish between firmware update packages applicable to different instantiations of the same downstream device (e.g. different instantiations are proxied by different FDs)
  - A mechanism to identify the component image that is to be transferred based on device identifier records. A device identifier record may be based on PCI IDs, SCSI ID, IEEE ID, or a vendor specific ID.

492 - A mechanism to determine that all similar downstream devices supported by an FDP are
493 to be updated using the same component image in a single transfer
494 - A mechanism to permit the UA to select the specific downstream device to be updated
495 using the component image from within a firmware update package
496
497 • Activation Requirements Gathering
498 - A mechanism to learn the activation requirements of the FD or downstream device
499 firmware components
500 - This will allow more timely and coordinated activation of all firmware components in the
501 system
502 - An FD may also be able to provide information about pending component images which
503 have been transferred through non-PLDM based transfer methods. For example a host
504 based update may have occurred but is in a pending state awaiting activation. In this
505 case, the FD may also be using other PLDM methods such as PLDM for Monitoring &
506 Control with a State Set sensor of ID 18 (Version) that indicates a version change was
507 detected.

508 Activation requirements for self-activation capable firmware devices or downstream devices shall specify
509 recovery times.

510 7.1 Firmware Update Concepts

511 A Firmware Device (FD) is the minimum hardware unit that the PLDM-based firmware update is applied
512 to and with which the Update Agent (UA) communicates to accomplish the update. The Firmware Update
513 Package for an FD may contain an individual component image or a group of component images which is
514 known as a component image set. This firmware update package is processed to update each firmware
515 component of the FD during the PLDM update.

516 A Downstream Device is optionally supported as an FD-attached entity that a FD can proxy firmware
517 update for. The downstream device does not directly communicate to the Update Agent, but the FD which
518 is acting as proxy can support firmware inventory and firmware update commands on the downstream
519 device's behalf. An Update Agent that performs firmware updates, will use similar but separate
520 sequences to update the FD itself or the downstream device attached to the FD. The method, protocols,
521 and behavior of how the FD communicates with the downstream device is outside the scope of this
522 specification. This specification defines requirements and behavior for the FD acting as a proxy.

523 Each type of FD has a globally unique identity which can be used to distinguish it from other types of FDs.
524 A device identifier record consisting of a set of device descriptors, which are typically based on industry
525 standard definitions, may be used to describe an FD type. For example, the descriptors for PCI devices
526 may include PCI Vendor ID and PCI Device ID.

527 Because an FD could be used in different instantiations (such as using the same device on an I/O
528 adapter vs. on a system planar), which may require different firmware loads, a corresponding more
529 specific set of device descriptors may be necessary to identify the type of FD intended for the update. For
530 example, for PCI devices the additional descriptors such as PCI Subsystem Vendor ID and PCI
531 Subsystem ID may be added to the identifier record used to match a firmware update package to an FD.

532 Component images that comprise the overall firmware update package each have a classification,
533 identifier, an optional component comparison stamp, and version.

534 - Classification: identifies the function type of the component image, such as UEFI driver, port controller
535 firmware, update SW, diagnostic code, firmware bundle, etc.

536 - Identifier: A unique value (per vendor) that distinguishes between component images which may have
537 identical classifications but contain different code images.

538 - Component Comparison Stamp: An optional vendor-assigned value that can be used to compare levels
539 between the firmware component within the FD and the component image within the firmware update

package. For example, an FD vendor might use a value for this field in the format of MajorMinorRevisionPatch where each subfield has a range of 0x00 to 0xFF. The component comparison stamp if implemented shall contain a value that can be compared to another component comparison stamp using an unsigned integer compare. Therefore when comparing component comparison stamps the lower value is down-level compared to the other when performing an unsigned integer comparison between the two.

- Version: Contains a string describing the component image version. The version string for the component image is provided by the FD vendor.

7.2 Update Agent

The Update Agent (UA) is a function that is present within a PLDM subsystem that has the ability to discover firmware devices and downstream devices which are capable of performing a PLDM firmware update and subsequently transfer one or more component images to the device. Only one UA function is supported within a given PLDM subsystem.

7.3 PLDM Firmware Update Packaging

The firmware update package provides the necessary information to be used with the PLDM Firmware Update commands.

To assist in performing an update over PLDM, the firmware update package shall contain a firmware package header describing the contents of the firmware update package. The header shall include (refer to Section 8 for details of the header structure):

1. A header info area describing the overall packaging version, date
2. Device identifier records to describe which FDs the update is intended for
3. Downstream Device identifier records to describe which downstream devices the update is intended for
4. Package contents information describing the component images contained within the package, including their classification, offset, size, and version
5. A checksum

7.4 Update Flow Overview – FD Update

The flow diagram example below describes the high level process of how the UA updates a FD. This flow occurs after the UA has determined which FD(s) the firmware update package is intended for. If there is an error or timeout whereby the entire firmware update process is canceled, then the UA may choose to reattempt the firmware update by sending another RequestUpdate command to the FD.

NOTE: A single FD is only permitted to have one update flow ongoing, while a UA may have multiple flows simultaneously in process if they are to multiple FDs.

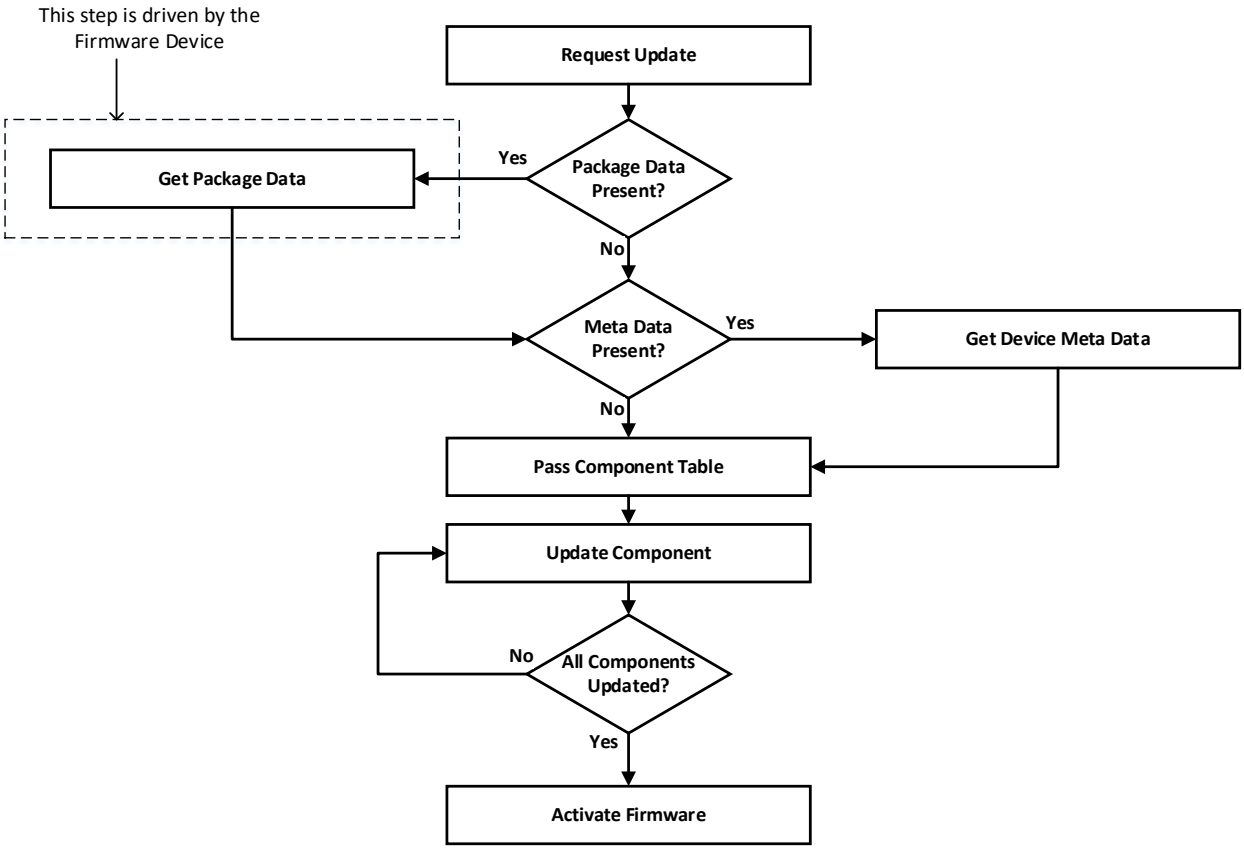


Figure 1 – High Level Firmware Update Flow

As shown in Figure 1, updating an FD is divided into these general steps.

1. To initiate a firmware update, the UA sends the PLDM command RequestUpdate to an FD. The FD replies with a response indicating whether it is available for firmware update. At this time, the FD is not aware of the specific component image version levels that the UA will attempt to transfer, only the component image set version is provided. The FD shall then enter an update mode that no longer permits another update request until the UA finishes or cancels the firmware update. During this firmware update mode, the device may or may not be able to provide normal service to the system depending on the capability of the device. The indication of this ability will be returned in the GetFirmwareParameters command.

2. If the firmware update package contains optional package data for the firmware device, then the UA shall transfer the package data to the FD prior to transferring component images. Refer to Section 8 for more details about the optional package data.

3. The UA may also optionally retrieve FD metadata which will be saved by the UA during the firmware update process and restored back to the FD after all component images have been transferred

4. The UA passes the component information table described in the firmware package header to the FD, which includes the identifier, component comparison stamp, classification, and version information for each of the applicable component images. This is performed by issuing one or more PassComponentTable PLDM commands.

5. The UA processes each of the applicable component images in the firmware update package one by one in the same sequence as is described in the firmware package header. The detailed steps of updating a component are described in section 7.6.

6. After all component images have been successfully transferred, verified and applied into the firmware device's non-volatile storage, the UA will send the ActivateFirmware command to the FD to finish the firmware update sequence. The FD can return a maximum activation time required to perform the operation. Upon receiving the ActivateFirmware command, if self-contained activation is supported and requested by the UA, the FD should immediately enable the new component images which were transferred to become the actively running code image. The FD will then exit from update mode at the conclusion of the activation. The FD may not be able to provide normal service when activating firmware (as the endpoint may require a restart). The UA periodically sends GetStatus to the FD within the maximum activation time to detect when the activation completes.

Note that for components which do not support self-contained activation, the ActivateFirmware command instructs the FD to perform FD-specific actions required to set the remaining updated firmware components into a 'pending activation' state. The newly transferred component images will then become the actively running code images upon external activation (such as a medium specific reset or a host reboot). Non-self-contained activation may also be supported through the activation pending component commands if the UA and FD support those optional commands.

7. The UA may send the CancelUpdate command at any time during the update process to the FD during firmware update, for example if an error is encountered. The FD will then exit update mode which completes the firmware update procedure. It is strongly recommended that the entire firmware update procedure is performed as a single sequence of events to avoid issues that may occur on the FD with partially updated firmware components.

8. If the UA is no longer able to communicate with the FD in order to cancel update mode, the FD itself shall provide an internal timer to exit from update mode if no commands are received. Refer to FD_T1 in section 7.12 of this document. If the FD had begun the apply or activate step, then it shall finish that operation before exiting from update mode, otherwise the FD should attempt to discard the component image and exit from update mode.

7.5 Update Flow Overview – Downstream Update

The flow diagram example below describes the high level process of how the UA updates a downstream device. This flow occurs after the UA has determined which downstream devices the firmware update is intended for. The UA will interact with the FDP which will act as proxy for the downstream device. If there is an error or timeout whereby the entire firmware update process is canceled, then the UA may choose to reattempt the firmware update by sending another RequestDownstreamDeviceUpdate command to the FDP.

NOTE: A Firmware Update Package may include component images for both the FDP device itself, as well as the downstream devices supported by the FDP. The UA must execute updates to the FDP firmware components and its supported downstream devices independently and complete one before the other is attempted. For example, if the FDP has one disk drive attached to it, and the Firmware Update Package has a component image for both the FDP and the disk drive, the UA must update one before the other. A single FDP is only permitted to have one update flow ongoing, while a UA may have multiple flows simultaneously in process if they are to multiple FDPs for separate downstream device updates.

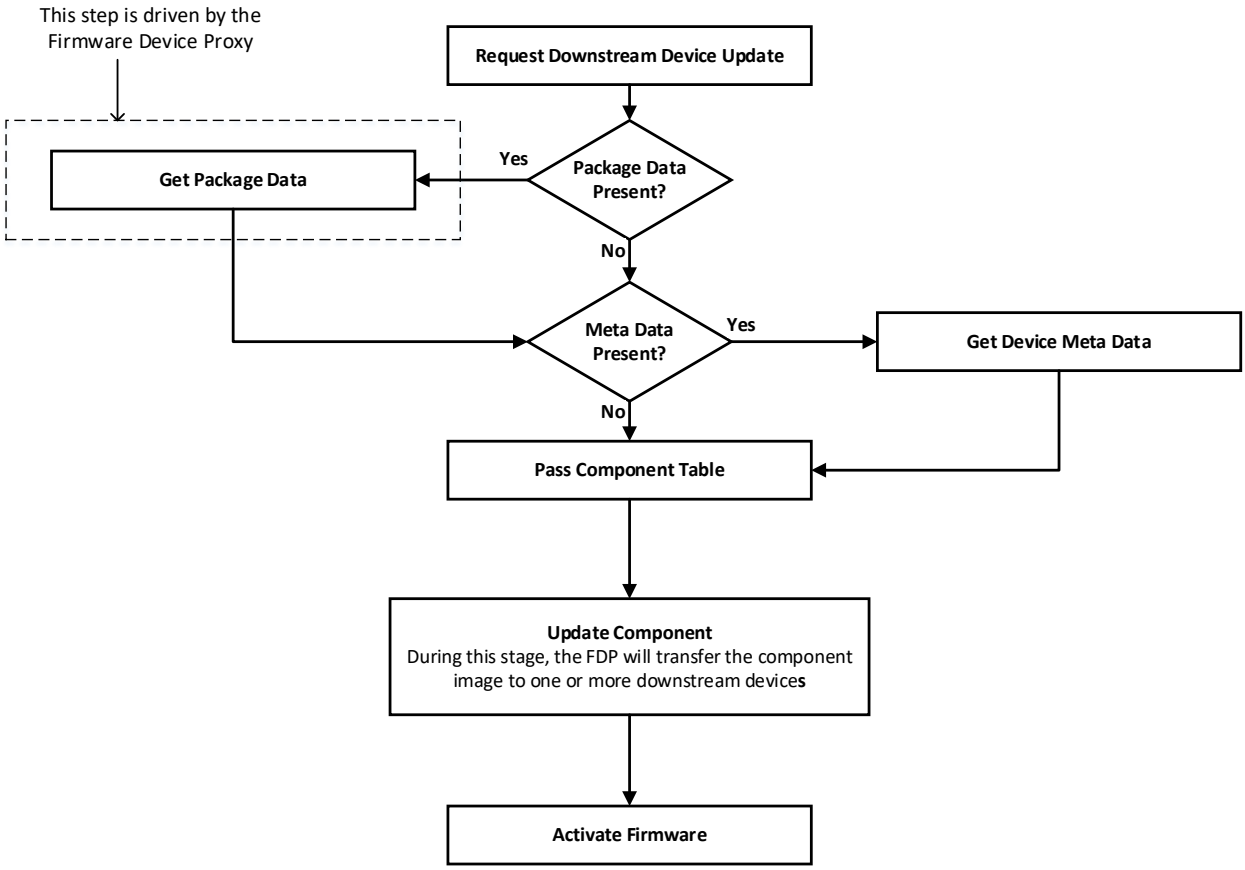


Figure 2 – High Level Firmware Update Flow for Downstream Devices

As shown in Figure 1, updating a downstream is divided into these general steps.

1. To initiate a downstream device firmware update, the UA sends the PLDM command RequestDownstreamDeviceUpdate to an FDP which is acting as a proxy for the downstream device. The FDP replies with a response indicating whether it is available for firmware update. The FDP shall then enter an update mode that no longer permits another update request until the UA finishes or cancels the firmware update. During this firmware update mode, both the FDP and/or the downstream device may or may not be able to provide normal service to the system depending on the capability of the device. The indication of this ability will be returned in the GetDownstreamFirmwareParameters command.

2. If the firmware update package contains optional package data for the downstream device, then the UA shall transfer the package data to the FDP prior to transferring component images. Refer to Section 8 for more details about the optional package data.

3. The UA passes the component information table described in the firmware package header to the FDP, which includes the identifier, component comparison stamp, classification, and version information for the applicable component image.

4. The UA will determine whether one or more downstream devices (of the same type - where all device descriptors match) and their firmware components will be updated with the component image. This is provided in the UpdateComponent command that is sent to the FDP.

5. After the component image has been successfully transferred, verified and applied into the downstream device's non-volatile storage, the UA will send the ActivateFirmware command to the FDP to finish the firmware update sequence for downstream devices. The FDP can return a maximum activation time required by the FDP and downstream device to perform the operation. Upon receiving the ActivateFirmware command, if self-contained activation is supported and requested by the UA, the FDP should immediately enable the new component images on the downstream devices which were transferred to become the actively running code image. The FDP will then exit from update mode at the conclusion of the activation. The FDP or downstream device may not be able to provide normal service when activating firmware (as the endpoint may require a restart). The UA periodically sends GetStatus to the FDP within the maximum activation time to detect when the activation completes.

Note that for downstream device firmware components which do not support self-contained activation, the ActivateFirmware command instructs the FDP to perform FDP-specific actions required to set the remaining updated firmware components into a 'pending activation' state on the downstream device. The newly transferred component images will then become the actively running code images upon external activation (such as a medium specific reset or a host reboot). Non-self-contained activation may also be supported through the activation pending component commands if the UA and FDP support those optional commands.

6. The UA may send the CancelUpdate command at any time during the update process to the FDP during firmware update, for example if an error is encountered. The FDP will then exit update mode which completes the firmware update procedure to the downstream device. It is strongly recommended that the entire firmware update procedure is performed as a single sequence of events to avoid issues that may occur on the FDP or downstream device with partially updated firmware components.

7. If the UA is no longer able to communicate with the FDP in order to cancel update mode, the FDP itself shall provide an internal timer to exit from update mode if no commands are received. Refer to FD_T1 in section 7.12 of this document. If the FDP had begun the apply or activate step, then it shall finish that operation before exiting from update mode, otherwise the FDP should attempt to discard the component image for the downstream device and exit from update mode.

7.6 Detailed Steps of Updating a Firmware Component

The steps below define transactions required to update one firmware component within a firmware device. If there is any error or timeout during the transfer of a component image, the timing specifications defined within DSP0240 shall be followed for command response timeouts and retries. In addition, specific PLDM Firmware Update timing specifications are defined in section 7.12 and shall be followed.

1. The UA sends the UpdateComponent command, providing component classification, component version, component size, and update options to begin the process of updating a specific firmware component.

2. The FD proceeds to request the component image, by sending one or more RequestFirmwareData commands to the UA. The request command specifies a component image portion to be transferred via the offset and length fields in the RequestFirmwareData command. The UA will validate the request, and if within the permitted range of the component image defined by the firmware package header and additional padding, generate a successful response containing the component image portion requested by the FD. Refer to Table 34 for details on the permitted range for the request.

   The size of the component image portion requested shall:

   - Be equal to or larger than the firmware update baseline transfer size

- Not exceed the MaximumTransferSize value received in the RequestUpdate command.
- Not require the UA to add an amount of padding bytes which is greater than the firmware update baseline transfer size.

After a successful transmission of RequestFirmwareData, the FD sends the next RequestFirmwareData command to get the next portion of the component image. This step iterates until the FD receives all data transfers that are required for updating the firmware component, and signals the end of component image transfer to the Update Agent by the TransferComplete command. The UA will then proceed to the verification phase. The TransferComplete command may also be used by the FD to signal the detection of an error condition that terminates the data transfer of the component image.

3. Upon completing the component image transfer, the FD sends the TransferComplete command and transitions to the VERIFY state to verify the payload transferred. The UA can optionally send the GetStatus command to query the completion status of the verification process asynchronously. The verify step may require a large amount of time depending on the FD and the operations it must perform to verify the firmware component.

4. Once the firmware component is verified as valid by FD-specific methods, the FD sends VerifyComplete command to the UA. The FD, upon sending the command, transitions to the APPLY state which applies the payload transferred into its non-volatile storage area. Note that some FDs may not have a separate apply step as the component image was being directly placed into the final memory destination in parallel while the component image was being requested. This can occur if the FD does not have a temporary memory location to store the transfer prior to committing the component image to the permanent memory location. In this case the FD shall report this auto-apply mode of operation to the UA via the GetFirmwareParameters command, and the FD would send an ApplyComplete command immediately after the VerifyComplete command.

   It is recommended that the FD temporarily disable any other management operations which may cause a reset of the device until this apply step is complete.

   The UA can optionally send the GetStatus command periodically to query the completion status of this step. The apply step may require a large amount of time depending on the FD and the operations it must perform to apply the firmware component.

   After component apply is complete, the FD may determine that the activation method for this firmware component is different than that reported previously in the GetFirmwareParameters command. This change in activation method shall be indicated in the ApplyComplete command. Upon completion of the apply step the FD sends the ApplyComplete command to the UA, and transitions to the READY XFER state upon receiving a successful response message from the UA.

5. If additional component images remain, the UA shall continue to the next component image by sending another UpdateComponent command. Each component image shall be transferred individually in the order which they were listed within the firmware update package.

6. Once all applicable component images have been transferred, the UA shall send ActivateFirmware, and can optionally request activation for all firmware components that indicated support for Self-Contained activation. Activation of firmware components which require a medium-specific reset, system reboot, or power cycle shall be initiated by higher level systems management software having a broader view of the overall system state. However, the ActivateFirmware command informs the FD to do any preparation necessary to use the newly transferred component images at the next activation event.

There are two additional commands which the UA can send to the FD during the update process.

1. The UA may send the CancelUpdateComponent command to cancel the update of the current component image being transferred. If the FD has currently requested a portion of component image data via the RequestFirmwareData command, the UA should first respond to any outstanding RequestFirmwareData commands received before sending its request to CancelUpdateComponent when possible. If the FD had begun the apply or activate step, then it shall finish that operation, otherwise the FD should attempt to discard the component image. This specification does not describe or provide guidance on a recovery procedure if the FD operation is affected by a partially transferred image. Upon receiving this command, the FD remains in update mode and is capable of receiving another UpdateComponent command.

2. The UA may send the CancelUpdate command to cancel the entire firmware update process. Upon receiving the command, the FD returns to the Idle state and exits from update mode. If the FD had begun the apply or activate step, then it shall finish that operation before exiting from update mode, otherwise the FD should attempt to discard the component image and exit from update mode. This specification does not describe or provide guidance on a recovery procedure if the FD operation is affected by a partially transferred image. After canceling the update, the FD may not be able to operate normally if only a portion of the firmware update has been completed.

It is strongly recommended that the entire firmware update procedure be performed as a single sequence of events and not cancelled by the UA.

Other timeouts or retries may occur and the timing specification defined within section 7.12 shall be followed.

Figure 3 shows the flow for updating a single firmware component.

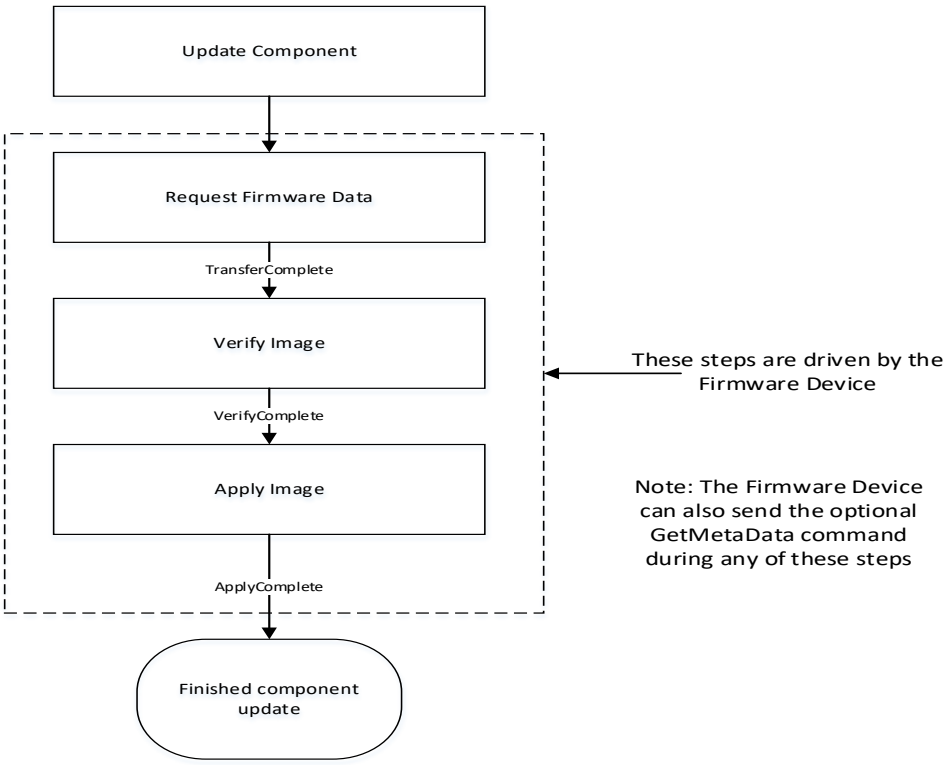


Figure 3 – Firmware Component Update Flow

7.7 Detailed Steps of Updating a Firmware Component – Downstream Update

The steps below define transactions required to update one firmware component within a downstream device. In order to perform the steps within this section, the UA will communicate to an FDP which is acting on behalf of the downstream device. If there is any error or timeout during the transfer of a component image, the timing specifications defined within DSP0240 shall be followed for command response timeouts and retries. In addition, specific PLDM Firmware Update timing specifications are defined in section 7.12 and shall be followed.

1. The UA sends the UpdateComponent command to the FDP, providing component classification, component version, component size, and update options to begin the process of updating the component image on the downstream device, only one component image can be supported on a downstream device. The UA can request for a single downstream device to be updated by the component image, or multiple downstream devices of the same type (where all device descriptors match).

2. The FDP proceeds to request the component image, by sending one or more RequestFirmwareData commands to the UA. The request command specifies a component image portion to be transferred via the offset and length fields in the RequestFirmwareData command. The UA will validate the request, and if within the permitted range of the component image defined by the firmware package header and additional padding, generate a successful response containing the component image portion requested by the FDP. Refer to Table 34 for details on the permitted range for the request.

   The size of the component image portion requested shall:
   - Be equal to or larger than the firmware update baseline transfer size
   - Not exceed the MaximumTransferSize value received in the RequestDownstreamDeviceUpdate command.

- Not require the UA to add an amount of padding bytes which is greater than the firmware update baseline transfer size.

After a successful transmission of RequestFirmwareData, the FDP sends the next RequestFirmwareData command to get the next portion of the component image. This step iterates until the FDP receives all data transfers that are required for updating the firmware component on the downstream device, and signals the end of component image transfer to the Update Agent by the TransferComplete command. The UA will then proceed to the verification phase. The TransferComplete command may also be used by the FDP to signal the detection of an error condition that terminates the data transfer of the component image.

Upon completing the component image transfer, the FDP sends the TransferComplete command and transitions to the VERIFY state to verify the payload transferred. The UA can optionally send the GetStatus command to query the completion status of the verification process asynchronously. The verify step may require a large amount of time depending on the FDP and the operations it must perform to verify the firmware component.

3. Once the firmware component is verified as valid by FDP-specific methods, the FDP sends VerifyComplete command to the UA. The FDP, upon sending the command, transitions to the APPLY state which applies the payload transferred into the downstream device's non-volatile storage area. Note that some FDPs may not have a separate apply step as the component image was being directly placed into the final memory destination on the downstream device in parallel while the component image was being requested. This can occur if the FDP or downstream device does not have a temporary memory location to store the transfer prior to committing the component image to the permanent memory location. In this case the FDP shall report this auto-apply mode of operation to the UA via the GetDownstreamFirmwareParameters command, and the FDP would send an ApplyComplete command immediately after the VerifyComplete command.

   It is recommended that the FDP temporarily disable any other management operations which may cause a reset of the device until this apply step is complete.

   The UA can optionally send the GetStatus command periodically to query the completion status of this step. The apply step may require a large amount of time depending on the FDP and the operations it must perform to apply the firmware component on the downstream device.

   After component apply is complete, the FDP may determine that the activation method for this firmware component is different than that reported previously in the GetDownstreamFirmwareParameters command. This change in activation method shall be indicated in the ApplyComplete command. Upon completion of the apply step the FDP sends the ApplyComplete command to the UA, and transitions to the READY XFER state upon receiving a successful response message from the UA.

4. The UA shall send ActivateFirmware, and can optionally request activation for the firmware component which indicated support for Self-Contained activation. Activation of firmware components which require a medium-specific reset, system reboot, or power cycle shall be initiated by higher level systems management software having a broader view of the overall system state. However, the ActivateFirmware command informs the FDP to do any preparation necessary to use the newly transferred component images at the next activation event.

There are two additional commands which the UA can send to the FDP during the update process.

1. The UA may send the CancelUpdateComponent command to cancel the update of the current component image being transferred. If the FDP has currently requested a portion of component image data via the RequestFirmwareData command, the UA should first respond to any

outstanding RequestFirmwareData commands received before sending its request to CancelUpdateComponent. If the FDP had begun the apply or activate step, then it shall finish that operation, otherwise the FDP should attempt to discard the component image. This specification does not describe or provide guidance on a recovery procedure if the FDP or downstream device operation is affected by a partially transferred image. Upon receiving this command, the FDP remains in update mode and is capable of receiving another UpdateComponent command.

2. The UA may send the CancelUpdate command to cancel the entire firmware update process. Upon receiving the command, the FDP returns to the Idle state and exits from update mode. If the FDP had begun the apply or activate step of an individual component image, then it shall finish that operation before exiting from update mode, otherwise the FDP should attempt to discard the component image and exit from update mode. This specification does not describe or provide guidance on a recovery procedure if the FDP or downstream device operation is affected by a partially transferred image. After canceling the update, the FDP may not be able to operate normally if only a portion of the firmware update has been completed.

It is strongly recommended that the entire firmware update procedure be performed as a single sequence of events and not cancelled by the UA.

Other timeouts or retries may occur and the timing specification defined within section 7.12 shall be followed.

Figure 3 shows the flow for updating a firmware component on one or more downstream devices

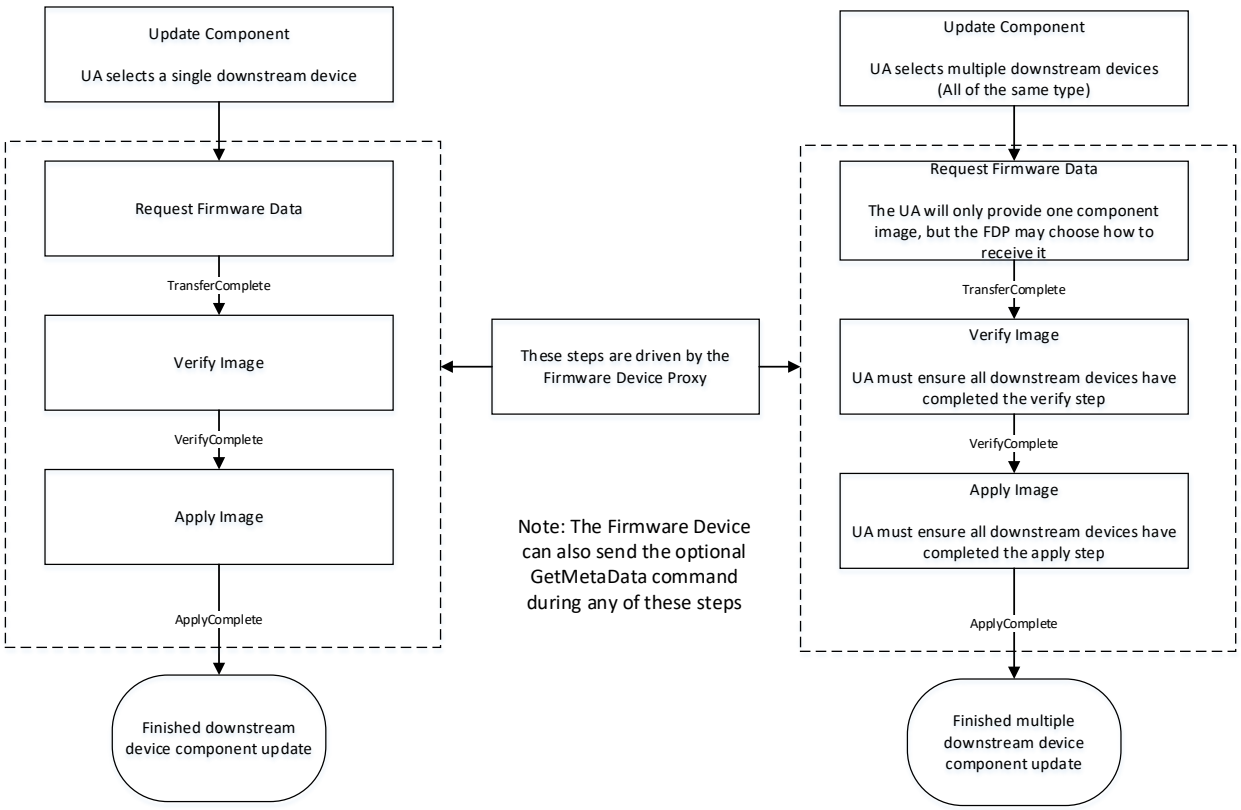


Figure 4 – Firmware Component Update Flow – Downstream Device

7.8 Firmware Update Baseline Transfer Size

The firmware update baseline transfer size is the minimum amount of bytes that can be requested through the RequestFirmwareData command by the FD. Both the FD and UA shall support the firmware update baseline transfer size. The UA can advertise a higher value which it may support as indicated by the MaximumTransferSize value in the RequestUpdate or RequestDownstreamDeviceUpdate command. The firmware update baseline transfer size is 32 bytes.

7.9 Firmware Component Authentication

The entire firmware update package could also be signed and authenticated by the UA prior to executing the PLDM Firmware update process, however this process is not within the scope of this specification and is not defined. A higher level entity that delivers the PLDM firmware update package to the Update Agent can add support for authentication.

Firmware components are required to be authenticated by the FD or downstream device through methods defined by the FD or downstream device manufacturer. It is recommended that the individual component images contain a signature which enhances the security of the firmware update. It is up to the FD or downstream device to decide what level of authentication will be performed by the FD or downstream device within the PLDM firmware update sequence during the verify process.

7.10 Type Code

Refer to DSP0245 for a list of PLDM Type Codes in use. This specification uses the PLDM Type Code 000101b as defined in DSP0245.

7.11 Error Completion Codes

PLDM completion codes for firmware update that are beyond the scope of PLDM_BASE_CODES in DSP0240 are defined in the list below. The usage of individual error completion codes are defined within each of the PLDM command sections.

Table 1 – PLDM Firmware Update Completion Codes

Value	Name	Returned By	Description
Various	PLDM_BASE_CODES	FD/FDP & UA	Refer to DSP0240 for a full list of PLDM Base Code Completion values that are supported.
0x80	NOT_IN_UPDATE_MODE	FD/FDP	Received PLDM firmware update command when the FD/FDP is not in update mode.
0x81	ALREADY_IN_UPDATE_MODE	FD/FDP	FD/FDP receives RequestUpdate or RequestDownstreamDeviceUpdate when it's already in update mode.
0x82	DATA_OUT_OF_RANGE	UA	The requested component image portion has an initial offset which is not contained within the image data, or the offset plus the length requested exceeds the range permitted by the UA.
0x83	INVALID_TRANSFER_LENGTH	UA	The length of the requested component image portion exceeds the MaximumTransferSize negotiated in the RequestUpdate or RequestDownstreamDeviceUpdate command, or is less than the firmware update baseline transfer size.

0x84	INVALID_STATE_FOR_COMMAND	FD/FDP	The FD/FDP is not in a state to expect this command.
0x85	INCOMPLETE_UPDATE	FD/FDP	One or more component transfers failed to complete.
0x86	BUSY_IN_BACKGROUND	FD/FDP	The FD/FDP is performing critical background task and cannot execute the command.
0x87	CANCEL_PENDING	UA	Sent by the UA when it receives a RequestFirmwareData command after sending a CancelUpdate or CancelUpdateComponent command.
0x88	COMMAND_NOT_EXPECTED	UA	Sent by the UA when it receives a command from the FD/FDP out of sequence from when it is expected.
0x89	RETRY_REQUEST_FW_DATA	UA	The Update Agent has requested a retry of the RequestFirmwareData command as it needs more time to retrieve the section of firmware to transfer.
0x8A	UNABLE_TO_INITIATE_UPDATE	FD/FDP	The FD/FDP is not able to enter into update mode to begin a transfer.
0x8B	ACTIVATION_NOT_REQUIRED	FD/FDP	The FD/FDP already has enabled the firmware components to become the active running image on the next external activation, or the firmware components are already activated.
0x8C	SELF_CONTAINED_ACTIVATION_ NOT_PERMITTED	FD/FDP	The firmware device or downstream device does not permit Self-Contained activation and returns this code when the UA requests a self-contained activation.
0x8D	NO_DEVICE_METADATA	FD/FDP	The FD/FDP has no meta data that must be retrieved by the UA prior to the start of the component image transfers.
0x8E	RETRY_REQUEST_UPDATE	FD/FDP	The FD/FDP has requested a retry of the RequestUpdate or RequestDownstreamDeviceUpdate command as it needs more time to prepare for a firmware update.
0x8F	NO_PACKAGE_DATA	UA	The Update Agent has no package data available for the firmware device
0x90	INVALID_TRANSFER_HANDLE	FD/FDP & UA	The data transfer handle requested was invalid
0x91	INVALID_TRANSFER_OPERATION _FLAG	FD/FDP & UA	The transfer operation flag used in the request was invalid
0x92	ACTIVATE_PENDING_IMAGE_NOT _PERMITTED	FD/FDP	The firmware device or downstream device does not support activating a pending component image or component image set
0x93	PACKAGE_DATA_ERROR	FD/FDP	The FD/FDP has received invalid Package Data and will not proceed with the firmware update.

0x94	NO_OPAQUE_DATA	UA	The Update Agent has no component opaque data available for the firmware device
0x95	UPDATE_SECURITY_REVISION_NOT_PERMITTED	FD	The FD/FDP does not support updating the security revision number of the component image
0x96	DOWNSTREAM_DEVICE_LIST_CHANGED	FD/FDP	The FD/FDP must end the transfer as one or more downstream devices were added or removed during the inventory transfer.

930

931 7.12 Timing Specification

932 Table 2 below defines timing values that are specific to this document. The table below defines the timing
933 parameters defined for the PLDM Firmware Update Specification. In addition, all timing parameters listed
934 in DSP0240 for command timeouts and number of retries shall also be followed. Figure 5 provides a
935 visual representation example of how the minimum and maximum timing parameters should be
936 implemented.

937 **Table 2 – Timing Specification**

Timing specification	Applicable to UA or FD	Symbol	Min	Max	Description
PLDM Base Timing	UA & FD	PNx PTx			Refer to DSP0240 for the details on these timing values which are applicable to PLDM message timeouts where a response is not received by the UA or FD/FDP after sending a request.
Number of request retries when a response is received that requires a retry	UA & FD	UAFD_T1	2		Total of three tries, minimum: the original try plus two retries.
Update mode idle timeout	FD	FD_T1	60 seconds	120 seconds	Amount of time before the FD/FDP shall exit from update mode if no command is received from the Update Agent when it's expected, during the firmware update process. For example, the FD shall wait a minimum of 60 seconds for the UA to send a PassComponentTable or UpdateComponent command.
Retry request for firmware data	FD	FD_T2	1 second	5 seconds	Amount of time for the FD/FDP to wait before resending a RequestFirmwareData command after receiving a RETRY_REQUEST_FW_DATA code from the UA.
Retry interval to send next cancel command	UA	UA_T1	500 milliseconds	5 seconds	Amount of time to wait before the UA sends an additional CancelUpdate or CancelUpdateComponent command.

Request firmware data idle timeout	UA	UA_T2	60 seconds	90 seconds	Amount of time for the Update Agent to cancel the component update if no command is received from the FD/FDP when it's expected, during the component image transfer stage. For example, the UA shall wait a minimum of 60 seconds for the FD to send another RequestFirmwareData command.
State change timeout	UA	UA_T3	180 seconds	-	Amount of time for the Update Agent to wait before canceling the component update if the ProgressPercent value in the GetStatus command remains unchanged.
Retry request for update	UA	UA_T4	1 second	5 seconds	Amount of time for the UA to wait before resending a RequestUpdate or RequestDownstreamDevice Update command after receiving a RETRY_REQUEST_UPDATE code from the FD/FDP.
Get Package Data timeout	UA	UA_T5	1 second	5 seconds	Amount of time for the UA to wait to receive the GetPackageData command if the FD/FDP indicated that it would send that command in the response to RequestUpdate or RequestDownstreamDeviceUpdate. The UA shall send CancelUpdate if this timer expires.
Complete Commands Timeout	UA	UA_T6	600 seconds		Amount of time for the UA to wait for a TransferComplete, VerifyComplete, or ApplyComplete command if the ProgressPercent value in the GetStatus command is set to 0x65 (not supported by FD/FDP). The UA uses this timeout to send a CancelUpdateComponent command upon receiving no further command from the FD/FDP after the last exchange and the FD/FDP does not support a ProgressPercent indication in GetStatus.

938

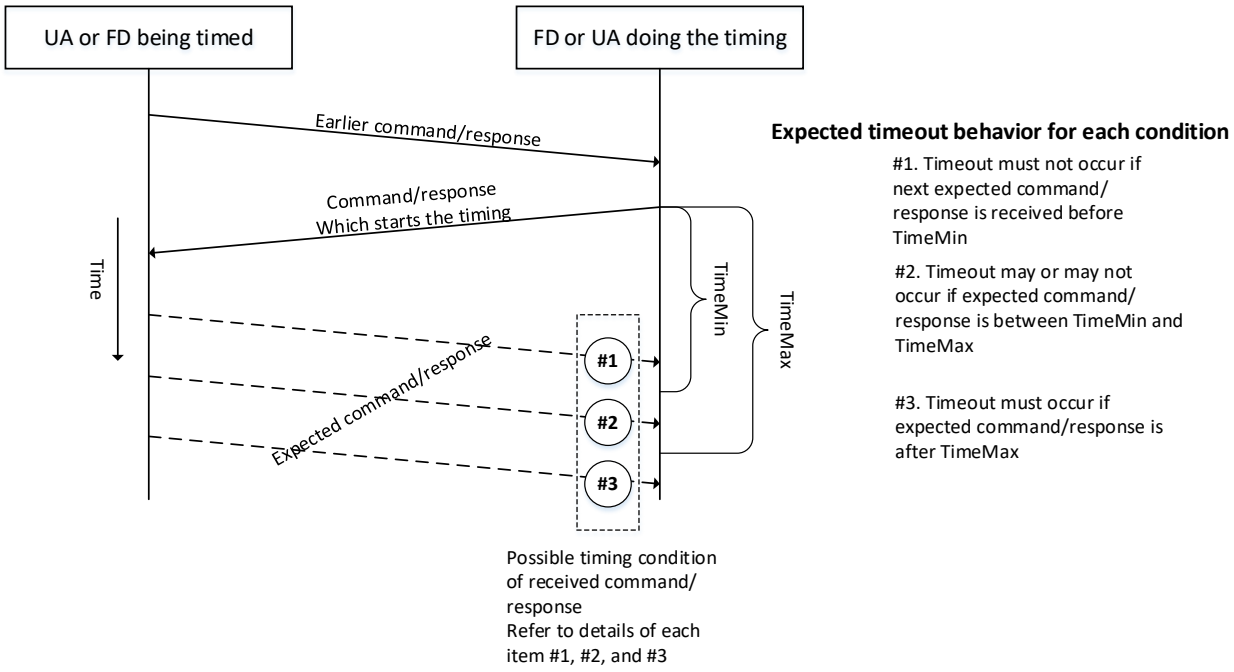


Figure 5 – Timeout Behavior Diagram

8 PLDM Firmware Update Package

A firmware update package that complies with the structure and requirements within this section shall be provided to the UA for processing and delivery of the component images to an FD using PLDM commands. The method of how the firmware update package is delivered to the UA is outside the scope of this specification.

The PLDM firmware update package contains two major sections; the firmware package header, and the firmware package payload.

The firmware package header is required to describe the firmware devices that the package is intended to update and the component images that the firmware update package contains.

The firmware update header supports the following:

- The firmware update package can be valid for multiple devices and allows for a method to describe each of the supported firmware devices or downstream devices.

  This is useful for the case when a device manufacturer has a family of different devices that use the same component images.

- The firmware update package can be specific to a particular instantiation of the same device

  This allows for the case such as where the planar implementation and/or one or more adapter implementations of the same device use different packages. In this case the device subsystem IDs could be used to differentiate between the two firmware devices or downstream devices.

- One to N explicit component images

The firmware update package can be used for a single monolithic image (component classification of Software Bundle) that contains 1 or more embedded code images. In this case it appears to the UA as if the package contains just one component image but is known by the FD or downstream device to contain multiple bundled code images. For an FD component image, it can also be used for multiple separate component images, each of which has a vendor-specific component identifier to distinguish between its different components. Up to 65535 components are supported.

A view which shows the entire firmware update package is below:

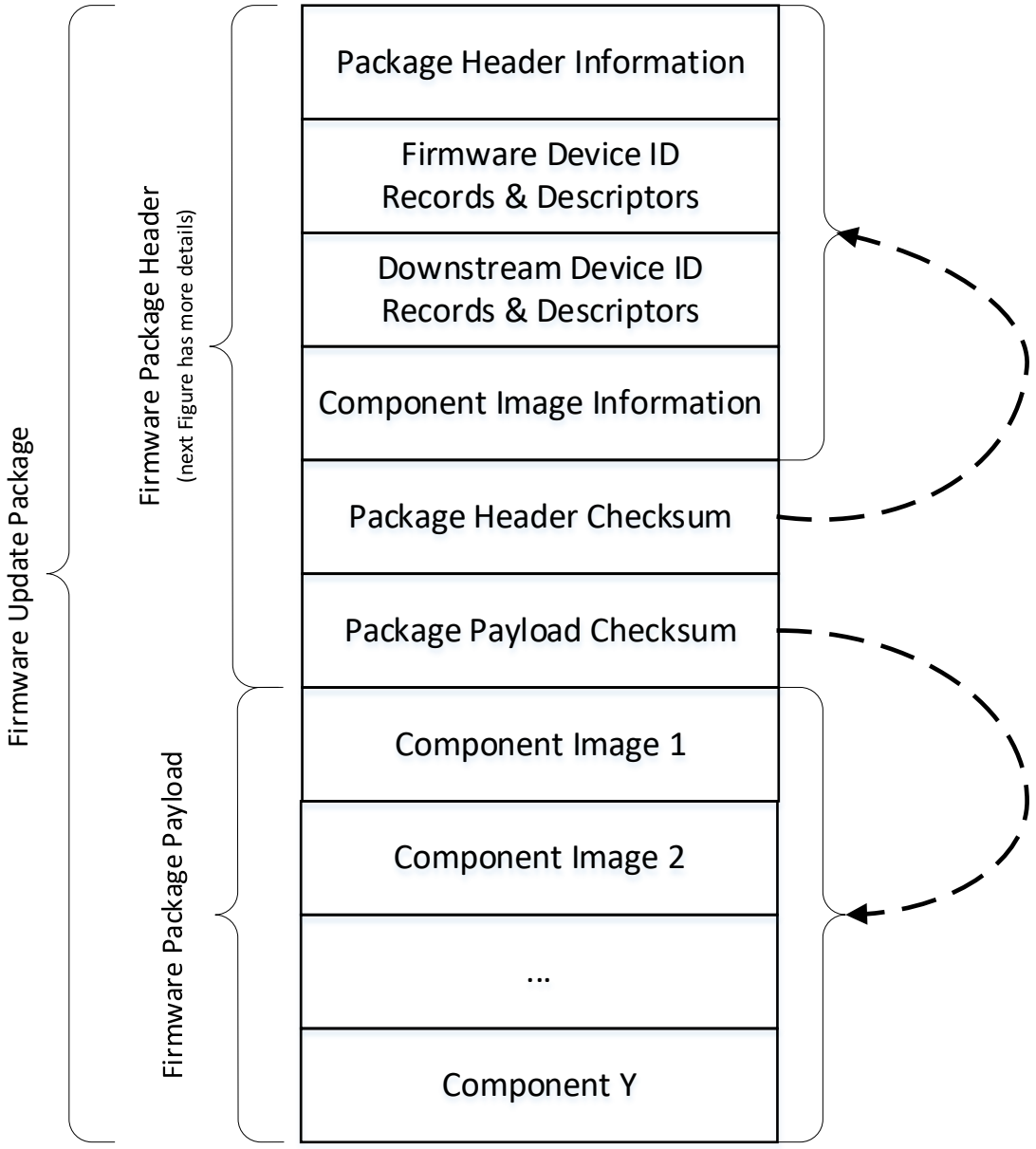


Figure 6 – PLDM Firmware Update Package

The following figure shows the structures within the firmware package header:

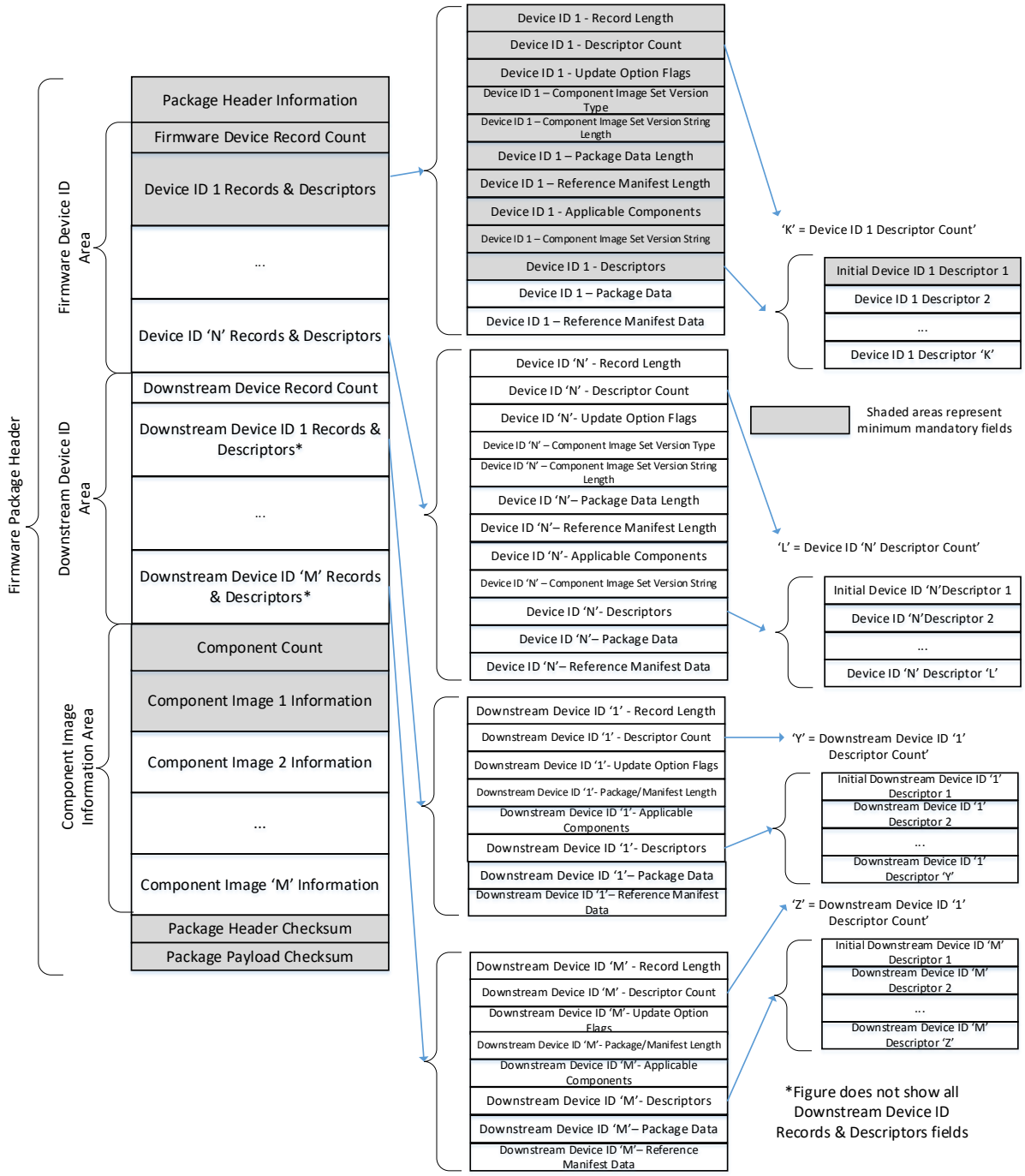


Figure 7 – PLDM Firmware Package Header Structure

The package header information fields contain details that describe the firmware update package and contains an identifier which the UA can use to identify that the contents within the package adhere to this specification.

The firmware device identification area is used to list the FDs that are supported by this firmware update package and the component images associated with the device. The order of the devices within the

Firmware Device Identification Area is of no significance and does not imply any order to the update of devices found to match.

The downstream device identification area is used to list the downstream devices that are supported by this firmware update package and the single component image associated with the device. The order of the devices within the Downstream Device Identification Area is of no significance and does not imply any order to the update of devices found to match.

The component image information area is used to describe the individual component images, the order in which they are transferred to the firmware device, and where each component image resides within the firmware update package.

The package header checksum field provides an integrity checksum for the entire firmware package header contents.

The package payload checksum field provides an integrity checksum for the entire firmware package payload contents. The firmware package payload contains the individual component images that can be transferred to the firmware devices. Prior to transferring the component images, the header shall be parsed by the UA to identify the following:

- Determine if the firmware update package is applicable for updating a specific FD or downstream device by comparing device identifier records in the package header to those obtained from the FD via the QueryDeviceIdentifiers or QueryDownstreamIdentifiers command.

- Locate the component image for each firmware component if multiple components are contained in the firmware update package. A bitmap of which packaged components are intended for which matched FDs or downstream devices is also contained in the header.

A firmware update package may contain one or more component images applicable to a single FD. The UA shall advertise each component image individually and shall transfer each of the component images, contained within the component image set, to the FD. The firmware package header provides the information to be able to identify a component by comparing its identifier value, along with additional information such as the component classification.

1011 **Table 3 – PLDM Firmware Package Header**

Package Header Information Byte ordering for applicable header fields is Little Endian per Section 5.2	
Type	**Definition**
UUID	**PackageHeaderIdentifier** Mandatory label which defines this object as a valid PLDM Firmware Update Package which includes a formatted header that complies to this specification. A value of 7B291C996DB64208801B02026E463C78 shall indicate that the update package format conforms to Version 1.3 (this version). A value of 3119CE2FE80A4A99AF6D46F8B121F6BF shall indicate that the update package format conforms to Version 1.2 A value of 1244D2648D7D4718A030FC8A56587D5A shall indicate that the update package format conforms to Version 1.1 A value of F018878CCB7D49439800A02F059ACA02 shall indicate that the update package format conforms to Version 1.0 UUID field is Big Endian. Refer to the PLDM Base Specification for field format definition.
uint8	**PackageHeaderFormatRevision** The revision number of the header structure itself. Updated when any field in the PLDM Firmware Update Header changes. This field shall be set to 0x04 for this version of the PLDM Firmware Update Header. (Adds support for Package Payload Checksum and Reference Manifest Data fields.) A value of 0x03 shall indicate that the package header format is described by DSP0267 version 1.2.x level. (Adds support for Component Opaque Data.) A value of 0x02 shall indicate that the package header format is described by DSP0267 version 1.1.x level. (Adds support for Downstream Devices.) A value of 0x01 shall indicate that the package header format is described by DSP0267 version 1.0.x level. All other values are Reserved.
uint16	**PackageHeaderSize** The count of all bytes in this header structure including the fields contained within the Package Header Information, Firmware Device Identification Area, Downstream Device Identification Area, Component Image Information Area, and the Package Checksum sections.
timestamp 104	**PackageReleaseDateTime** The date and time in which this package was released. Refer to the PLDM Base Specification for field format definition.
uint16	**ComponentBitmapBitLength** The number of bits that will be used to represent the bitmap in the ApplicableComponents field for a matching device. The value shall be a multiple of 8 and be large enough to contain a bit for each component in the package.
enum8	**PackageVersionStringType** The type of string used in the PackageVersionString field. Refer to Table 33 for values.
uint8	**PackageVersionStringLength** The length, in bytes, of the PackageVersionString field.
Variable	**PackageVersionString** Package version information, up to 255 bytes. Contains a variable type string describing the version of this firmware update package.

Firmware Device Identification Area	
Type	**Definition**
uint8	**DeviceIDRecordCount** The count of firmware device ID records that are defined within this package. Each record consists of information about the firmware device including; the component image set that is applicable for transfer to the device, record descriptors, and optional package data. Each record contains a set of identifier descriptors and a component image bitmap indicating applicable firmware components in the package intended for the FD. If all descriptors contained in one of the records matches the record of identifiers returned from the FD via the QueryDeviceIdentifiers command then this package is applicable to the FD.
Variable	**FirmwareDeviceIDRecords** Refer to Table 4 for details of this field. Contains a record, a set of descriptors, and optional package data for each firmware device within the count provided from the DeviceIDRecordCount field.
Downstream Device Identification Area	
Type	**Definition**
uint8	**DownstreamDeviceIDRecordCount** The count of downstream device ID records that are defined within this package. Each record consists of information about the downstream device including; the component image set that is applicable for transfer to the device, record descriptors, and optional package data. Each record contains a set of downstream identifier descriptors and a component image bitmap indicating the applicable firmware component in the package intended for the downstream device which will be proxied by an FD. If all descriptors contained in one of the records matches the record of identifiers returned for the downstream device from the FD proxy via the QueryDownstreamIdentifiers command then this package is applicable to the Downstream device.
Variable	**DownstreamDeviceIDRecords** Refer to Table 5 for details of this field. Contains a record, a set of descriptors, and optional package data for each downstream device within the count provided from the DownstreamDeviceIDRecordCount field.
Component Image Information Area	
Type	**Definition**
uint16	**ComponentImageCount** Count of individual separately defined component images contained within this firmware update package.
Variable	**ComponentImageInformation** Refer to Table 6 for details of this field. Contains details for each component image contained within this firmware update package.
Package Header Checksum	
Type	**Definition**
uint32	**PackageHeaderChecksum** The integrity checksum of the PLDM Package Header. It is calculated starting at the first byte of the PLDM Firmware Update Header and includes all bytes of the package Header structure except for the bytes in this field and the PLDMFWPackagePayloadChecksum field For this specification, CRC-32 algorithm with the polynomial x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1 (same as the one used by IEEE 802.3) shall be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first.

PLDM Firmware Package Payload Checksum	
Type	**Definition**
uint32	**PLDMFWPackagePayloadChecksum** The integrity checksum of the PLDM Package Payload. It is calculated starting at the first byte immediately following this field and includes all bytes of the firmware package payload structure, including any padding of bytes between component images. For this specification, the CRC-32 algorithm with the polynomial x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1 (same as the one used by IEEE 802.3) shall be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first.

1012 The contents of the FirmwareDeviceIDRecords field is described in the following table.

1013 **Table 4 – Firmware Device ID Record**

Individual Firmware Device ID Record (this section is repeated for each Firmware Device ID)	
Type	**Definition**
uint16	**RecordLength** The total length in bytes for this record. The length shall include the RecordLength, DescriptorCount, DeviceUpdateOptionFlags, ComponentImageSetVersionStringType, ComponentSetVersionStringLength, FirmwareDevicePackageDataLength, ApplicableComponents, ComponentImageSetVersionString, RecordDescriptors, and FirmwareDevicePackageData fields.
uint8	**DescriptorCount** The number of descriptors included within the RecordDescriptors field for this record.
bitfield32	**DeviceUpdateOptionFlags** 32 bit field, each bit represents an update option. [31:1] – Reserved [0] – Continue component updates after failure If set, the UA shall attempt to update any remaining components after an individual component update fails as the FD will remain in the Update mode. This includes continuing after a non-zero ComponentResponseCode is received from the FD in the PassComponentTable command response.
enum8	**ComponentImageSetVersionStringType** The type of string used in the ComponentImageSetVersionString field. Refer to Table 33 for values.
uint8	**ComponentImageSetVersionStringLength** The length, in bytes, of the ComponentImageSetVersionString.
uint16	**FirmwareDevicePackageDataLength** The length in bytes of the FirmwareDevicePackageData field. If no data is provided in the firmware update package for the Firmware Device described by this portion of the header, then this length field should be set to 0x0000.
uint32	**ReferenceManifestLength** The length in bytes of the ReferenceManifestData field. If no data is provided in the firmware update package for the Reference Manifest described by this portion of the header, then this length field should be set to 0x00000000.

Variable Bitfield	**ApplicableComponents** The size of this bitfield is based on the value contained in the ComponentBitmapBitLength field. Bitmap of which firmware components are applicable to FDs which match this Device Identifier record. A set bit N indicates the Nth (0-based) component in the payload (which is described by the Nth entry in the component information area of the package header) is applicable to this device. Since the Component Bitmap Bit Length field (a multiple of 8) may contain bit positions not associated with any component (if the number of components is not a multiple of 8), those bit positions will contain 0 and are located in the high order bit positions within the bitfield.
Variable	**ComponentImageSetVersionString** Component Image Set version information, up to 255 bytes. Contains a variable type string describing the version of the set of component images which are applicable to the firmware device indicated in this device ID record.
Variable	**RecordDescriptors** Refer to Table 7 for details of these fields and the values that can be selected.
Variable	**FirmwareDevicePackageData** An optional data field that can be provided within the firmware update package which the UA shall transfer to the FD during the firmware update process. The UA has no knowledge of what data is contained within this field, and will simply pass the contents of this field when the FD requests it via the GetPackageData command response. If the FirmwareDevicePackageDataLength field is set to 0x0000 then this field shall contain no data and is zero bytes in length.
Variable	**ReferenceManifestData** An optional data field that can be provided within the firmware update package to contain a Reference Manifest for the firmware update. If present, the Reference Manifest shall describe the firmware update provided by this package. The UA can use the data in this field as references for the firmware version provided in this package. The UA shall not transfer this data to the FD. The format of this field shall be a Standard Body or Vendor-Defined Header, as shown in Table 10 which identifies the entity that defined the Reference Manifest format, followed by the Reference Manifest data. If the ReferenceManifestLength field is set to 0x00000000, then this field shall contain no data and is zero bytes in length.

A firmware device record shall have at least one descriptor, but typically will have additional descriptors that the UA will use to match against a FD. Each descriptor is comprised of three fields: (1) Type (2) Length (3) Value. The initial descriptor is restricted to one of five types, while additional descriptors can choose from a larger range of type values including a vendor defined type. Refer to Table 7 for more details.

The contents of the DownstreamDeviceIDRecords field is described in the following table.

1020 **Table 5 – Downstream Device ID Record**

Individual Downstream Device ID Record (this section is repeated for each Downstream Device ID)	
Type	**Definition**
uint16	**DownstreamDeviceRecordLength** The total length in bytes for this record. The length shall include the DownstreamDeviceRecordLength, DownstreamDeviceDescriptorCount, DownstreamDeviceUpdateOptionFlags, DownstreamDeviceSelfContainedActivationMinVersionStringType, DownstreamDeviceSelfContainedActivationMinVersionStringLength, DownstreamDevicePackageDataLength, DownstreamDeviceApplicableComponents, DownstreamDeviceSelfContainedActivationMinVersionString, DownstreamDeviceSelfContainedActivationMinVersionComparisonStamp, DownstreamDeviceRecordDescriptors, and DownstreamDevicePackageData fields.
uint8	**DownstreamDeviceDescriptorCount** The number of descriptors included within the DownstreamDeviceRecordDescriptors field for this record.
bitfield32	**DownstreamDeviceUpdateOptionFlags** 32 bit field, each bit represents an update option. [31:1] – Reserved [0] – Downstream Device can support self-contained activation with minimal version level defined by DownstreamDeviceSelfContainedActivationMinVersion fields
enum8	**DownstreamDeviceSelfContainedActivationMinVersionStringType** The type of string used in the DownstreamDeviceSelfContainedActivationMinVersionString field. Refer to Table 33 for values. If bit 0 of DownstreamDeviceUpdateOptionFlags is set to 0 then this field is set to 0
uint8	**DownstreamDeviceSelfContainedActivationMinVersionStringLength** The length, in bytes, of the DownstreamDeviceSelfContainedActivationMinVersionString field. If bit 0 of DownstreamDeviceUpdateOptionFlags is set to 0 then this field is set to 0x0
uint16	**DownstreamDevicePackageDataLength** The length in bytes of the DownstreamDevicePackageData field. If no data is provided in the firmware update package for the Downstream Device described by this portion of the header, then this length field should be set to 0x0000.
uint32	**DownstreamDeviceReferenceManifestLength** The length in bytes of the DownstreamDeviceReferenceManifestData field. If no data is provided in the firmware update package for the Reference Manifest described by this portion of the header, then this length field should be set to 0x00000000.
Variable Bitfield	**DownstreamDeviceApplicableComponents** The size of this bitfield is based on the value contained in the ComponentBitmapBitLength field. For Downstream Devices only one component images shall be selected. Bitmap of which firmware components are applicable to Downstream Devices which match this Downstream Device Identifier record. A set bit N indicates the Nth (0-based) component in the payload (which is described by the Nth entry in the component information area of the package header) is applicable to this device. Since the Component Bitmap Bit Length field (a multiple of 8) may contain bit positions not associated with any component (if the number of components is not a multiple of 8), those bit positions will contain 0 and are located in the high order bit positions within the bitfield.

Variable	**DownstreamDeviceSelfContainedActivationMinVersionString** Downstream Device self contained activation minimum version string, up to 255 bytes. Contains a variable type string describing the minimum version that must be the currently active image on the downstream device which can support a self-contained activation. If bit 0 of DownstreamDeviceUpdateOptionFlags is set to 0, then this field does not exist.
Variable	**DownstreamDeviceSelfContainedActivationMinVersionComparisonStamp** Downstream Device self contained activation minimum comparison stamp. Contains a variable type string describing the minimum version that must be the currently active image on the downstream device which can support a self-contained activation. If bit 0 of DownstreamDeviceUpdateOptionFlags is set to 0 then this field does not exist. If bit 0 of DownstreamDeviceUpdateOptionFlags is set to 1 then this field is a uint32 value.
Variable	**DownstreamDeviceRecordDescriptors** Refer to Table 7 for details of these fields and the values that can be selected.
Variable	**DownstreamDevicePackageData** An optional data field that can be provided within the firmware update package which the UA shall transfer to the downstream device via the FDP which will act as a proxy during the firmware update process. The UA has no knowledge of what data is contained within this field, and will simply pass the contents of this field when the FDP requests it via the GetPackageData command response. If the DownstreamDevicePackageDataLength field is set to 0x0000 then this field shall contain no data and is zero bytes in length.
Variable	**DownstreamDeviceReferenceManifestData** An optional data field that can be provided within the firmware update package to contain a Reference Manifest for the Downstream Device firmware update. If present, the Reference Manifest shall describe the Downstream Device firmware update provided by this package. The UA can use the data in this field as references for the firmware version provided in this package. The UA shall not transfer this data to the FDP or the Downstream Device. The format of this field shall be a Standard Body or Vendor-Defined Header, as shown in Table 10 which identifies the entity that defined the Reference Manifest format, followed by the Reference Manifest data. If the DownstreamDeviceReferenceManifestLength field is set to 0x00000000, then this field shall contain no data and is zero bytes in length.

A downstream device record shall have at least one descriptor, but may have additional descriptors that the UA will use to match against a downstream device. Each descriptor is comprised of three fields: (1) Type (2) Length (3) Value. The initial descriptor is restricted to one of seven types, while additional descriptors can choose from a larger range of type values including a vendor defined type. Refer to Table 7 for more details.

The contents of the ComponentImageInformation field is described in the following table.

Table 6 – Component Image Information

Individual Component Image Information (repeated for each component image)	
Type	**Definition**
uint16	**ComponentClassification** FD vendor selected value to indicate specific FD component. Values for this field are aligned with the Value Map from CIM_SoftwareIdentity.Classifications. Refer to Table 32 for values. If ComponentClassification = 0xFFFF, this indicates the component image is for a downstream device

uint16	**ComponentIdentifier** FD vendor selected unique value to distinguish between component images. If ComponentClassification = 0xFFFF to state this component image is for a downstream device, then this field shall be set to 0xFFFF in the package header.
uint32	**ComponentComparisonStamp** When ComponentOptions bit 1 is set, this field shall contain a FD or downstream device vendor selected value to use as a comparison value in determining if a firmware component is down-level or up-level. For the same component identifier, the greater of two component comparison stamps is considered up-level compared to the other when performing an unsigned integer comparison. FD vendors should choose the value for the comparison stamp in a manner that permits interim component versions such as patch releases. For example, a value for this field may follow the format of MajorMinorRevisionPatch where each subfield has a range of 0x00 to 0xFF. When ComponentOptions bit 1 is not set, this field should use the value of 0xFFFFFFFF.
bitfield16	**ComponentOptions** [15:4] – reserved [3] – Security revision update support When set, this component image supports the ability to update the security revision. The UpdateComponent command provides an option for the FD/FDP to request deferral of the update to the security revision number until after activation [2] – Component Image has corresponding Component Opaque Data When set, this component image has additional data contained within the firmware update package header. The value in ComponentOpaqueDataLength must be greater than zero, and the actual data must be provided in the ComponentOpaqueData field. [1] – Use Component Comparison Stamp When set, this bit indicates to the UA that the ComponentComparisonStamp field should be used for comparing this component against the component currently installed within the FD or downstream device. If this bit is not set, the UA can only use the ComponentVersionString information which may not provide a direct comparison method to determine whether the component is higher or lower than one which is currently installed within the FD. [0] - Force Update When set, this bit indicates to the UA that it should request a comparison override (update the firmware component even if the update would take the component to a lower or equal component comparison stamp, or version string, than is currently active) in the UpdateComponent command for this component.
bitfield16	**RequestedComponentActivationMethod** Provides the ability for the firmware update package to request an activation method that the UA should use for the component images being updated. The UA would use the information from this field, along with the activation methods supported by the firmware device and/or downstream device directly to determine the appropriate method for activation of the new code. Set each requested activation method to 1b (multiple choices are possible). [15:7] – Reserved [6] – Management Controller (MC) reset [5] - AC power cycle [4] - DC power cycle [3] - System reboot [2] - Medium-specific reset [1] - Self-Contained (can be performed upon transmission of ActivateFirmware command) [0] - Automatic (becomes active as the Apply completes, or as download completes if the FD performs an auto-apply)

uint32	**ComponentLocationOffset** Offset in Bytes from byte 0 of the package header to where the component image begins.
uint32	**ComponentSize** Size in Bytes of the Component image.
enum8	**ComponentVersionStringType** The type of string used in the ComponentVersionStringField. Refer to Table 33 for values.
uint8	**ComponentVersionStringLength** The length, in bytes, of the ComponentVersionString.
Variable	**ComponentVersionString** Component version information up to 255 bytes. Contains a variable type string describing the component version.
uint32	**ComponentOpaqueDataLength** The length in bytes of the ComponentOpaqueData field. If no data is provided in the firmware update package for the Component Image described by this portion of the header, then this length field shall be set to 0x00000000.
Variable	**ComponentOpaqueData** An optional data field that can be provided within the firmware update package which the UA shall transfer to the FD/FDP during the firmware update process. The UA has no knowledge of what data is contained within this field, and will simply pass the contents of this field when the FD/FDP requests it via the GetComponentOpaqueData command response. If the ComponentOpaqueDataLength field is set to 0x0000 then this field contains no data and is zero bytes in length.

1028 The content of the RecordDescriptors field is described in the following table.

1029 **Table 7 – Descriptor Definition**

Initial Descriptor (This first initial descriptor (Type, Length, and Value) is mandatory)	
Type	**Definition**
uint16	**InitialDescriptorType** Indicates the type of the Initial descriptor. Refer to Table 8 for possible values. The initial descriptor for a device shall be defined by one of the highlighted rows identified in Table 8. Some descriptors can be used as the initial descriptor for either a Firmware, Downstream, or Individual Device, whereas some may only be used as the initial descriptor for a Downstream or Individual Device. If the FD uses Vendor Defined values as part of its implementation of this specification (for example to provide a vendor defined error code or component classification), then the initial descriptor shall be set to either PCI Vendor ID or IANA Enterprise ID. If the downstream or individual device uses Vendor Defined values as part of its implementation of this specification (for example to provide a vendor defined error code or component classification), then the initial descriptor shall be set to either PCI Vendor ID, IANA Enterprise ID, IEEE Assigned Company ID or SCSI Vendor ID.
uint16	**InitialDescriptorLength** Indicates the length, in bytes, of the InitialDescriptorData field. Refer to Table 8 for possible values.
Variable	**InitialDescriptorData** Payload containing the identifier value for the initial descriptor. Refer to Table 8 for details.

Optional Additional Descriptors (repeated for each additional descriptor) For each additional descriptor three fields are provided (Type, Length, Value)	
Type	**Definition**
uint16	**AdditionalDescriptorType** Indicates the type of the additional descriptor. Refer to Table 8 for possible values.
uint16	**AdditionalDescriptorLength** Indicates the length, in bytes, of the AdditionalDescriptorIdentifierData field. Refer to Table 8 for possible values.
Variable	**AdditionalDescriptorIdentifierData** Payload containing the identifier value for the additional descriptors. Refer to Table 8 for details.

The following table provides a list of available descriptor types that can be used by the firmware package header and FD or downstream devices. When the FD or downstream device is a PCI device, there are up to four descriptors that are mandatory to be implemented.

Table 8 – Descriptor Identifier Table

Any one of the highlighted rows can be used for the Initial Device Descriptor			
Type	**Length**	**Initial Descriptor Usage**	**Value**
0x0000 – PCI Vendor ID	2 bytes	Firmware, Downstream, or Individual Device	PCI Vendor ID assigned to the device vendor. If the FD or downstream device is a PCI device, this descriptor shall be the initial descriptor.
0x0001 – IANA Enterprise ID	4 bytes	Firmware, Downstream, or Individual Device	IANA Enterprise ID assigned to the device vendor.
0x0002 – UUID	16 bytes	Firmware, Downstream, or Individual Device	UUID assigned to the device. Refer to PLDM Base Specification for UUID format. Version 1 format is recommended.
0x0003 – PnP Vendor ID	3 bytes	Firmware, Downstream, or Individual Device	PnP Vendor ID, in ASCII characters, assigned to the device vendor. Refer to the PnP & ACPI Registry for more details. http://www.uefi.org/PNP_ACPI_Registry
0x0004 – ACPI Vendor ID	4 bytes	Firmware, Downstream, or Individual Device	ACPI Vendor ID, in ASCII characters, assigned to the device vendor. Refer to the PnP & ACPI Registry for more details. https://uefi.org/PNP_ACPI_Registry
0x0005 – IEEE Assigned Company ID	3 bytes	Downstream or Individual Device Only	IEEE Company ID, assigned to the downstream device
0x0006 – SCSI Vendor ID	8 bytes	Downstream or Individual Device Only	SCSI Vendor ID, in ASCII characters, assigned to the downstream device

0x0100 – PCI Device ID	2 bytes	Cannot be used as an initial descriptor	PCI Device ID assigned by the device vendor. If the FD or downstream device is a PCI device, this descriptor shall be provided in the QueryDeviceIdentifiers/QueryDownstreamIdentifiers command response and shall also be used in the Descriptor Definition of the PLDM Firmware Package Header.
0x0101 – PCI Subsystem Vendor ID	2 bytes	Cannot be used as an initial descriptor	PCI Subsystem Vendor ID assigned to the device vendor. If the FD or downstream device is a PCI device, this descriptor shall be provided in the QueryDeviceIdentifiers/QueryDownstreamIdentifiers command response. This descriptor can optionally be used in the Descriptor Definition of the PLDM Firmware Package Header.
0x0102 – PCI Subsystem ID	2 bytes	Cannot be used as an initial descriptor	PCI Subsystem Device ID assigned by the device vendor. If the FD or downstream device is a PCI device, this descriptor shall be provided in the QueryDeviceIdentifiers/QueryDownstreamIdentifiers command response. This descriptor can optionally be used in the Descriptor Definition of the PLDM Firmware Package Header.
0x0103 – PCI Revision ID	1 byte	Cannot be used as an initial descriptor	PCI Revision ID assigned by the device vendor. This descriptor is optional for a PCI device. If this descriptor is used, the FD/FDP can report multiple PCI Revision ID values using multiple descriptors, where the highest value is the actual value but lesser values can be listed for firmware component compatibility. This descriptor can optionally be used in the Descriptor Definition of the PLDM Firmware Package Header. If any one of them match to the firmware package header Device ID record descriptors, then the component image is applicable to the FD/FDP device.
0x0104 – PnP Product Identifier	4 bytes	Cannot be used as an initial descriptor	PnP Product Identifier, in ASCII characters, assigned to the device vendor. Refer to the PnP & ACPI Registry for more details. http://www.uefi.org/PNP_ACPI_Registry
0x0105 – ACPI Product Identifier	4 bytes	Cannot be used as an initial descriptor	ACPI Product Identifier, in ASCII characters, assigned by the device vendor. Refer to the PnP & ACPI Registry for more details. http://www.uefi.org/PNP_ACPI_Registry
0x0106 – ASCII Model Number (Long String)	40 bytes	Cannot be used as an initial descriptor	Downstream Device Model number, in ASCII characters, assigned by the downstream device vendor. String shall be padded with ASCII null characters 0x00 bytes if needed to use the entire fixed length of the field.
0x0107 – ASCII Model Number (Short String)	10 bytes	Cannot be used as an initial descriptor	Downstream Device Model number, in ASCII characters, assigned by the downstream device vendor. String shall be padded with ASCII null characters 0x00 bytes if needed to use the entire fixed length of the field.

0x0108 – SCSI Product ID	16 bytes	Cannot be used as an initial descriptor	Downstream Device SCSI Product ID, in ASCII characters, assigned by the downstream device vendor. String shall be padded with ASCII null characters 0x00 bytes if needed to use the entire fixed length of the field.
0x0109 – UBM Controller Device Code	4 bytes	Cannot be used as an initial descriptor	The silicon identity device code of a Universal Backplane Management (UBM) controller
0x010A – IEEE EUI-64 ID	8 bytes	Cannot be used as an initial descriptor	Downstream Device IEEE EUI-64 global identifier, assigned by the downstream device vendor
0x010B – PCI Revision ID Range	2 bytes	Cannot be used as an initial descriptor	PCI Revision ID range assigned by the device vendor. This descriptor is optional for a PCI device when the initial descriptor is 0x0000. If this descriptor is used, a range of PCI Revision IDs can be described for a FD or downstream device where all PCI Revision IDs between the first value and last value (inclusive) are to be used to check for a match with a device identifier. The FD or downstream device identifier only needs to be matched to one of the PCI Revision IDs within the range. Byte 0 – First PCI Revision ID value Byte 1 – Last PCI Revision ID value
0xFFFF – Vendor Defined	Variable	Cannot be used as an initial descriptor	See Table 9 If the Device or package header uses a Vendor Defined value then the initial descriptor shall be set to either PCI Vendor ID, IANA Enterprise ID, IEEE Assigned Company ID, or SCSI Vendor ID

The following table provides details for the value field of a vendor defined descriptor.

Table 9 – Vendor Defined Descriptor Value Definition

Type	Definition
enum8	**VendorDefinedDescriptorTitleStringType** The type of string used in the VendorDefinedDescriptorTitleString field. Refer to Table 33 for values
uint8	**VendorDefinedDescriptorTitleStringLength** The length, in bytes, of the VendorDefinedDescriptorTitleString.
Variable	**VendorDefinedDescriptorTitleString** Vendor Defined Descriptor information up to 255 bytes. Contains a variable type string describing the Vendor's descriptor for the FD.
Variable	**VendorDefinedDescriptorData** Vendor-specific descriptor value. Value will be treated as binary data by the UA.

The following table provides details for the Standards Body or Vendor-defined header (SVH), which identifies the entity that defines the format for a given reference manifest. Note, if the reference manifest format in question is defined by a standards body, the SVH header does not require the use of the `VendorID` field. Instead, the `SVHID` field would be set to the ID of the standards body, `VendorIDLen` would be set to `0`, and `VendorID` would be absent. A standards body, registry, or vendor that defines a payload format should also define the values to use in the SVH header.

1042 **Table 10 – Standards Body or Vendor-defined Header (SVH) Table**

Type	Length	Value
enum8	1	**SVHID** Shall be one of the values in the SVHID column of Table 11
uint8	1	**VendorIDLen** Shall be the Length in bytes of the VendorID field. If the format of the given reference manifest is specified by a standards body or registry itself, this field shall be 0. Otherwise, if the format of the given reference manifest is specified by an organization that is identified on the vendor ID list indicated in the 'SVHID' field, this field shall be the length indicated in the "VendorIDLen" column of Table 11 for the respective SVHID .
Variable	VendorIDLen	**VendorID** If VendorIDLen is greater than zero, this field shall be the ID of the vendor corresponding to the SVHID field. Otherwise, this field shall be absent.

1043 The following table provides details for the SVHID field in the Standards Body or Vendor-defined header.

1044 **Table 11 – Registry or Standards Body Identifier Table**

SVHID	VendorIDLen (bytes)	Registry or Standards Body Name	Description
0x00	0	DMTF	DMTF does not have a Vendor ID registry.
0x01	2	TCG	Vendor is identified by using TCG Vendor ID Registry. For extended algorithms, see TCG Algorithm Registry.
0x02	2	USB	Vendor is identified by using the vendor ID assigned by USB.
0x03	2	PCI-SIG	Vendor is identified using PCI-SIG Vendor ID.
0x04	4	IANA	The Private Enterprise Number (PEN) assigned by the Internet Assigned Numbers Authority (IANA) identifies the vendor.
0x05	4	HDBaseT	Vendor is identified by using HDBaseT HDCD entity.
0x06	2	MIPI	The Manufacturer ID assigned by MIPI identifies the vendor.
0x07	2	CXL	Vendor is identified by using CXL vendor ID.
0x08	2	JEDEC	Vendor is identified by using JEDEC vendor ID.
0x09	0	VESA	For fields and formats defined by the VESA standards body, there is no Vendor ID registry.
0x0A	Variable	IANA CBOR	The CBOR Tag Registry that identifies the format of the element, as assigned by the Internet Assigned Numbers Authority (IANA). The encoding of the CBOR tag indicates the length of the tag. When a CBOR Tag is used with a Standards Body or Vendor-defined Header, the VendorIDLen field shall be set to the length of the encoded CBOR tag, followed by the data payload, which starts with an encoded CBOR tag.

8.1 Package to Firmware Device Association

The UA can associate a given firmware update package to all applicable FDs by using the following steps:

FOR each FD that supports PLDM for Firmware Update

Retrieve Firmware Device ID records via the QueryDeviceIdentifiers command

MATCH = FALSE; Start at First Firmware Device ID Record in the package header

WHILE ((MATCH==FALSE) AND (Firmware Device ID Record(s) remain in package))

Read Firmware Device ID Record from Package Header

IF all Firmware Device ID Record descriptors match FD descriptors

MATCH = TRUE; Selected Record = Current Record; Break;

Move to next Firmware Device ID Record in package header

Note that all descriptors in a package Firmware Device ID Record shall match those returned by the FD but not vice-versa (the FD may return more descriptors than are indicated in the firmware package header Firmware Device ID record). Some descriptors may have more than one value from the FD, for example PCI Revision ID, if multiple descriptors are provided by the FD, then just one that matches the descriptor in the package Firmware Device ID Records is a match.

Each FD that generated a match can accept components from the firmware update package.

8.2 Package to Downstream Device Association

The UA can associate a given firmware update package to all applicable downstream devices by using the following steps:

FOR each FDP that supports downstream devices which support PLDM for Firmware Update

Retrieve Downstream Device identifier records via the QueryDownstreamIdentifiers command

MATCH = FALSE; Start at First Downstream Device ID Record in the package header

WHILE ((MATCH==FALSE) AND (Downstream Device ID Record(s) remain in package))

Read Downstream Device ID Record from Package Header

IF all Downstream Device ID Record descriptors match FDP descriptors

MATCH = TRUE; Selected Record = Current Record; Break;

Move to next Downstream Device ID Record in package header

Note that all descriptors in a package Downstream Device ID Record shall match those returned by the FDP but not vice-versa (the FDP may return more descriptors than are indicated in the firmware package header Downstream Device ID record). Some descriptors may have more than one value from the FDP, for example PCI Revision ID, if multiple descriptors are provided by the FDP, then just one that matches the descriptor in the package Downstream Device ID Records is a match.

Each FDP that generated a match can accept components from the firmware update package.

8.3 Individual Firmware Device Package

An individual firmware device package complies with the structure and requirements within this section. This package is not directly used by the UA for processing and delivery of the component images to an FD using PLDM commands. The individual firmware device package can be used by the creator of the PLDM Firmware Update Package, as it is intended to provide information for a single device or similar device family that uses the same component image, while the PLDM Firmware Update Package can support a large number of FDs and DDs. For example, downstream devices that do not directly support PLDM commands could provide an individual firmware device package for the PLDM Firmware Package to be created by the FDP manufacturer. This individual firmware device package provides the necessary content, both header information and the component images, in order for them to be incorporated into the PLDM Firmware Package.

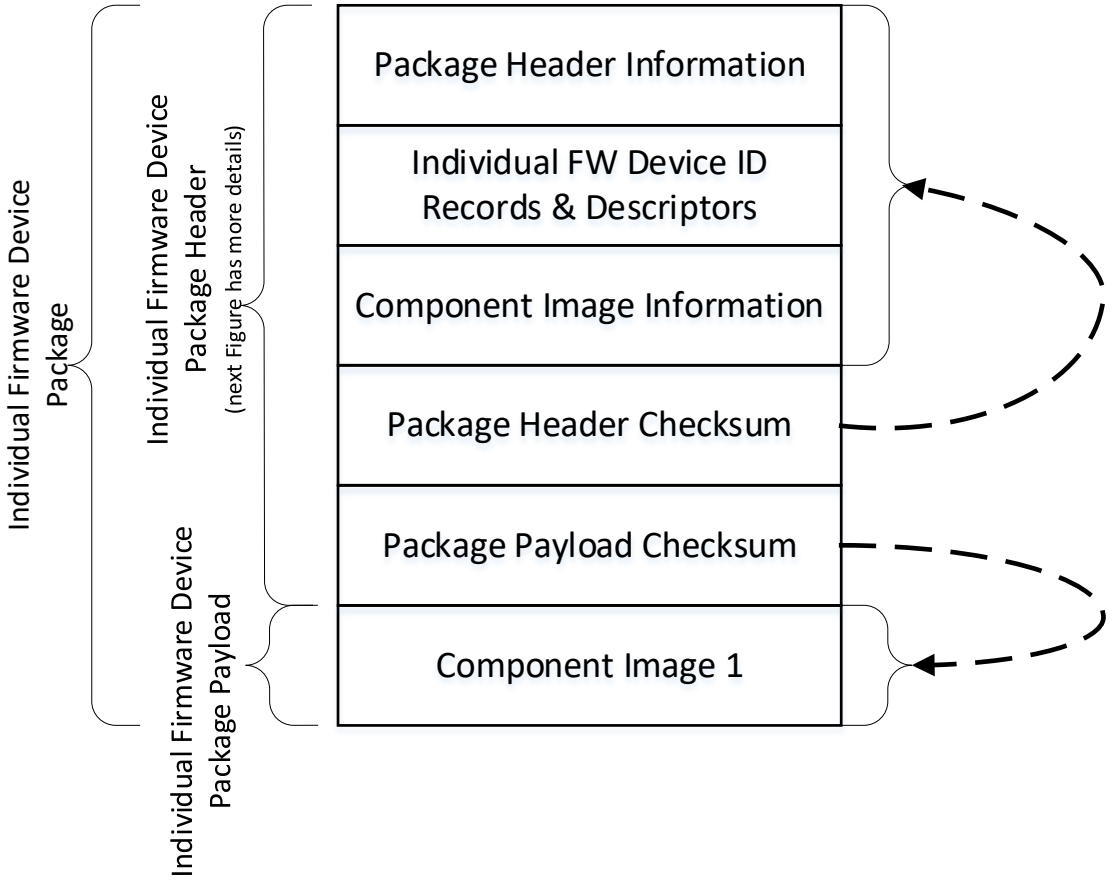
The specific method of how the individual firmware device package is incorporated into the PLDM Firmware Package is outside the scope of this specification.

The individual firmware device package contains two major sections; the individual firmware device package header, and the individual firmware device package payload.

The individual firmware device package supports the following:

- Header information to describe the capabilities and descriptors for a single device, or common device family
- One component image

A view which shows the entire individual firmware device package is below:



1100

1101 **Figure 8 – Individual Firmware Device Package**

1102
1103

The following figure shows the structures within the individual firmware device package header:

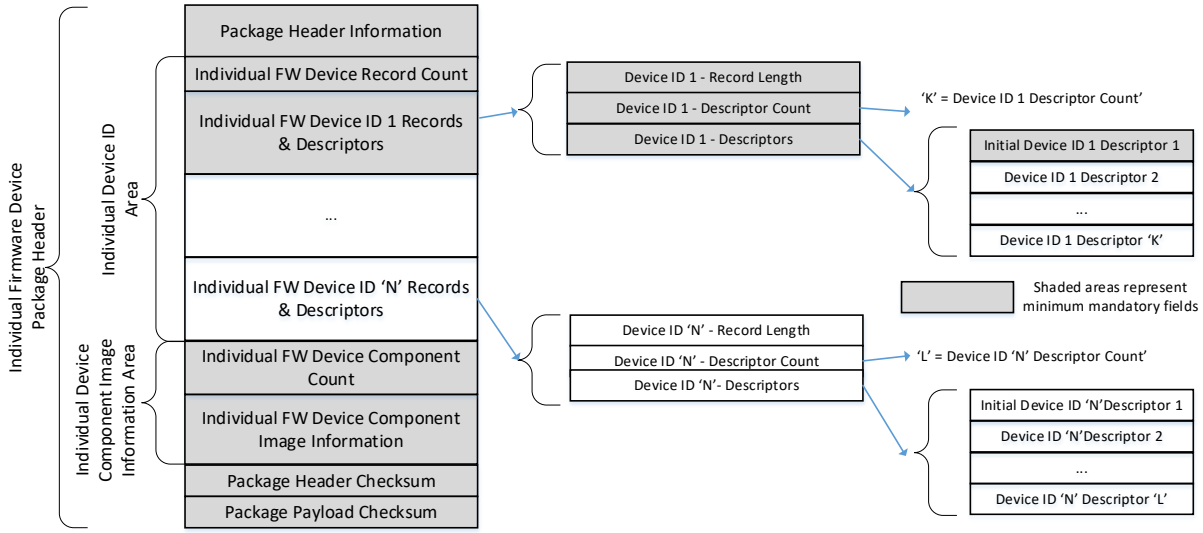


Figure 9 – Individual Firmware Device Package Header Structure

The individual firmware device package header information fields contain details that describe the individual firmware device package and contain an identifier which the creator of the PLDM Firmware Update Package can use to identify that the contents within the package adhere to this specification.

The individual firmware device identification area is used to list the single or common set of devices that are supported by this individual firmware device package and the single component images associated with the device. The order of the devices within the Individual Firmware Device Identification Area is of no significance and does not imply any order to the update of devices found to match.

The component image information area is used to describe the single individual firmware device component image, and where the individual firmware device component image resides within the individual firmware device package.

The package header checksum field provides an integrity checksum for the entire individual firmware device package header contents.

The package payload checksum field provides an integrity checksum for the entire individual firmware device package payload contents.

The individual firmware device package payload contains the single individual firmware device component image that support the devices described by the package.

1125 **Table 12 – Individual Firmware Device Package Header**

Individual Firmware Device Package Header Information Byte ordering for applicable header fields is Little Endian per Section 5.2	
Type	**Definition**
UUID	**IndividualFWDevicePackageHeaderIdentifier** Mandatory label which defines this object as a valid Individual Firmware Device Package which includes a formatted header that complies to this specification. A value of C7845F74CB69404AAE1A81A2609FA8A8 shall indicate that the update package format conforms to Version 1.3 (THIS VERSION) UUID field is Big Endian. Refer to the PLDM Base Specification for field format definition.
uint8	**IndividualDevicePackageHeaderFormatRevision** The revision number of the header structure itself. Updated when any field in the PLDM Firmware Update Header changes. This field shall be set to 0x01 for this version of the Individual Firmware Device Package Header. All other values are Reserved.
uint16	**IndividualDevicePackageHeaderSize** The count of all bytes in this header structure including the fields contained within the Individual Firmware Device Package Header Information, Individual Firmware Device Identification Area, Individual Firmware Device Component Image Information Area, and the Individual Firmware Device Package Checksum sections.
Individual Firmware Device Identification Area	
Type	**Definition**
uint8	**IndividualFWDeviceDeviceIDRecordCount** The count of downstream device ID records that are defined within this package. Each record consists of information about the downstream device including; the component image set that is applicable for transfer to the device, record descriptors, and optional package data.
Variable	**IndividualFWDeviceIDRecords** Refer to Table 13 for details of this field. Contains a record, a set of descriptors, and optional package data for each individual firmware device within the count provided from the IndividualFWDeviceIDRecordCount field.
Individual Firmware Device Component Image Information Area	
Type	**Definition**
uint16	**IndividualFWDeviceComponentImageCount** Field shall be set to 0x0001 A single component image is supported by the Individual Firmware Device Package.
Variable	**IndividualFWDeviceComponentImageInformation** Refer to Table 14 for details of this field. Contains details for the component image contained within this individual firmware device package.
Individual Firmware Device Package Header Checksum	
Type	**Definition**

uint32	**IndividualFWDevicePackageHeaderChecksum** The integrity checksum of the Individual Firmware Device Package Header. It is calculated starting at the first byte of the Individual Firmware Device Package Header and includes all bytes of the package Header structure except for the bytes in this field, and the IndividualFWDevicePackagePayloadChecksum field. For this specification, CRC-32 algorithm with the polynomial x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1 (same as the one used by IEEE 802.3) shall be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first.
Individual Firmware Device Package Payload Checksum	
Type	**Definition**
uint32	**IndividualFWDevicePackagePayloadChecksum** The integrity checksum of the Individual Firmware Device Package Payload. It is calculated starting at the first byte immediately following this field and includes all bytes of the individual firmware device package payload structure, including any padding of bytes before the component image. For this specification, CRC-32 algorithm with the polynomial x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1 (same as the one used by IEEE 802.3) shall be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first.

The contents of the IndividualFWDeviceIDRecords field is described in the following table.

Table 13 – Individual Firmware Device ID Record

Individual Firmware Device ID Record (this section is repeated for each Individual Firmware Device ID)	
Type	**Definition**
uint16	**IndividualFWDeviceRecordLength** The total length in bytes for this record. The length shall include the IndividualFWDeviceRecordLength, IndividualFWDeviceDescriptorCount, and IndividualFWDeviceRecordDescriptors.
uint8	**IndividualFWDeviceDescriptorCount** The number of descriptors included within the IndividualFWDeviceRecordDescriptors field for this record.
Variable	**IndividualFWDeviceRecordDescriptors** Refer to Table 7 for details of these fields and the values that can be selected.

An individual firmware device record shall have at least one descriptor, but typically will have additional descriptors. Each descriptor is comprised of three fields: (1) Type (2) Length (3) Value. The initial descriptor is restricted to one of three types, while additional descriptors can choose from a larger range of type values including a vendor defined type. Refer to Table 7 for more details.

The contents of the IndividualFWDeviceComponentImageInformation field is described in the following table.

Table 14 – Individual Firmware Device Component Image Information

Individual Firmware Component Image Information	
Type	**Definition**
uint32	**IndividualFWDeviceComponentLocationOffset** Offset in Bytes from byte 0 of the individual firmware device package header to where the component image begins.

uint32	**IndividualFWDeviceComponentSize** Size in Bytes of the Component image.
enum8	**IndividualFWDeviceComponentVersionStringType** The type of string used in the IndividualFWDeviceComponentVersionString field. Refer to Table 33 for values.
uint8	**IndividualFWDeviceComponentVersionStringLength** The length, in bytes, of the IndividualFWDeviceComponentVersionString field.
Variable	**IndividualFWDeviceComponentVersionString** Component version information up to 255 bytes. Contains a variable type string describing the component version.

9 Operational Behaviors

This clause describes the operating states of the FD.

9.1 State Definitions

The following states are required to be implemented by the FD.

- **IDLE**
  - IDLE is the default state in which the firmware device shall always start after an initialization. In this state the FD is not performing any firmware update actions as it has not received a RequestUpdate or RequestDownstreamDeviceUpdate command from the UA.
- **LEARN COMPONENTS**
  - After receiving the RequestUpdate or RequestDownstreamDeviceUpdate command, the FD moves to this state while waiting to receive the PassComponentTable command from the UA. The FD will then learn the size, identifier, component comparison stamp, classification and version of the component images the UA intends to send.
- **READY XFER**
  - After learning the component image information, the FD moves to this state to wait for the command initiating a component image transfer. This state is re-entered after each component image is transferred, verified and applied. The FD remains in this state after all firmware components have been applied as it waits for an activation command.
- **DOWNLOAD**
  - After receiving the command to update a firmware component, the FD moves to this state to begin requesting the transfer of portions of the component image from the UA. When an entire component image has been transferred, the UA is informed and the FD moves to the VERIFY state.
- **VERIFY**
  - In this state the FD performs a validation check of the firmware component, it is up to the FD to determine the method used for verification of the code image. Upon successful verification, the FD informs the UA and moves to the APPLY state.
- **APPLY**
  - In this state the FD writes the verified code image to the non-volatile storage area that will contain the code image within the device. When completed, the FD moves to the READY XFER state
- **ACTIVATE**

- The activation request from the UA occurs after all component images have been transferred, verified and applied. If requested, the FD performs immediate activation of the firmware components which have been described as supporting the 'self-contained' activation method. The FD also enables all other newly transferred code images to become the actively running firmware on the next initialization. After activation the FD moves to the IDLE state.

9.2 State Machine

The below table describes the operating states, responses, and transitions between states that the FD shall implement. The transition to the next state occurs after the FD performs the response action. In cases where the FD is sending a command to the UA, the transition does not occur until the UA successfully acknowledges the command (i.e., with a corresponding response and CompletionCode value of 0). Five commands; GetFirmwareParameters, QueryDeviceIdentifiers, QueryDownstreamDevices, QueryDeviceIdentifiers, and GetDownstreamFirmwareParameters are considered 'inventory' type commands and can be sent by the UA to the FD in any state. In addition, the GetStatus command may also be sent from the UA to the FD in any state.

Table 15 – Firmware Device State Machine

Current State	Trigger	Response	Next State
IDLE	RequestUpdate	Success	LEARN COMPONENTS
	RequestUpdate	Unable to Initiate Update or Retry Request Update	IDLE
	RequestDownstreamDeviceUpdate	Success	LEARN COMPONENTS
	RequestDownstreamDeviceUpdate	Unable to Initiate Update or Retry Request Update	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	IDLE
	QueryDownstreamDevices	Success with Downstream Devices	IDLE
	QueryDownstreamIdentifiers	Success with Downstream Identifiers	IDLE
	GetFirmwareParameters	Success with firmware info	IDLE
	GetDownstreamFirmwareParameters	Success with Downstream Device firmware info	IDLE
	GetStatus	Success with info	IDLE
	ActivatePendingComponentImageSet	Success	IDLE
	ActivatePendingComponentImage	Success	IDLE
	UpdateSecurityRevision	Success	IDLE
	UpdateSecurityRevision	Error with UPDATE_SECURITY_REVISION_NOT_PERMITTED	IDLE
	Any other command	Not in Update Mode	IDLE
LEARN COMPONENTS	FD_T1 timeout waiting for next command or response to GetPackageData	None	IDLE

	GetPackageData	Success	LEARN COMPONENTS
	GetPackageData with no package data	Error with NO_PACKAGE_DATA	LEARN COMPONENTS
	GetDeviceMetaData	Success	LEARN COMPONENTS
	GetDeviceMetaData with package data error	Error with PACKAGE_DATA_ERROR	LEARN COMPONENTS
	PassComponentTable with valid TransferFlag set to Start or Middle	Success	LEARN COMPONENTS
	PassComponentTable with valid TransferFlag set to End or StartAndEnd	Success	READY XFER
	PassComponentTable with invalid TransferFlag	Error CompletionCode	LEARN COMPONENTS
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	LEARN COMPONENTS
	QueryDownstreamDevices	Success with Downstream Devices	LEARN COMPONENTS
	QueryDownstreamIdentifiers	Success with Downstream Identifiers	LEARN COMPONENTS
	GetFirmwareParameters	Success with firmware info	LEARN COMPONENTS
	GetDownstreamFirmwareParameters	Success with Downstream Device firmware info	LEARN COMPONENTS
	GetStatus	Success with info	LEARN COMPONENTS
	Any other Update command	Invalid State Machine	LEARN COMPONENTS
READY XFER	FD_T1 timeout waiting for next command	None	IDLE
	RequestUpdate	Already In Update Mode	READY XFER
	GetFirmwareParameters	Success with firmware info	READY XFER
	GetDownstreamFirmwareParameters	Success with Downstream Device firmware info	READY XFER
	UpdateComponent with invalid or unsupported parameters	Non-zero ComponentCompatibilityResponseCode response	READY XFER
	UpdateComponent with supported and acceptable parameters	Success	DOWNLOAD
	GetMetaData	Success	READY XFER
	ActivateFirmware with self-contained activation requested after all expected components have completed transfer, verify and apply	Success with Activation Delay Time	ACTIVATE

	ActivateFirmware without self-contained activation requested after all expected components have completed transfer, verify and apply	Success	ACTIVATE → IDLE (FD moves through ACTIVATE step to IDLE)
	ActivateFirmware prior to all expected components completed	Incomplete Update response	READY XFER
	ActivateFirmware	No components required activation and ACTIVATION_NOT_REQUIRED is returned	ACTIVATE → IDLE (FD moves through ACTIVATE step to IDLE)
	ActivateFirmware	Self-Contained activation requested but not permitted	READY XFER
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	READY XFER
	QueryDownstreamDevices	Success with Downstream Devices	READY XFER
	QueryDownstreamIdentifiers	Success with Downstream Identifiers	READY XFER
	GetStatus	Success indicating READY XFER state	READY XFER
	Any other Update command	Invalid State Machine	READY XFER
DOWNLOAD	FD_T1 timeout waiting for response to RequestFirmwareData	None	IDLE
	Ready to request next component image portion	Send RequestFirmwareData command	DOWNLOAD
	Receive RequestFirmwareData response with image portion	Process data	DOWNLOAD
	All necessary data received and processed for this component	Send TransferComplete command with successful TransferResult	VERIFY
	Corrupt data received	Send TransferComplete command with failed TransferResult	DOWNLOAD
	Error response to RequestFirmwareData	Send TransferComplete command with failed TransferResult	DOWNLOAD
	Retry response to RequestFirmwareData	Delay, then send RequestFirmwareData command for same component image portion as prior request)	DOWNLOAD
	CancelUpdateComponent	Success	READY XFER
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	DOWNLOAD
	QueryDownstreamDevices	Success with Downstream Devices	DOWNLOAD
	QueryDownstreamIdentifiers	Success with Downstream Identifiers	DOWNLOAD
	GetFirmwareParameters	Success with firmware info	DOWNLOAD
	GetDownstreamFirmwareParameters	Success with Downstream Device firmware info	DOWNLOAD

	GetMetaData	Success	DOWNLOAD
	GetComponentOpaqueData	Success with Opaque Data	DOWNLOAD
	GetComponentOpaqueData	No Opaque Data	DOWNLOAD
	GetStatus while downloading	Download in progress	DOWNLOAD
	GetStatus after successful download	Download successful	DOWNLOAD
	Any other command	Invalid State Machine	DOWNLOAD
VERIFY	GetStatus while verifying	Verification in progress	VERIFY
	GetStatus after successful verify	Verification successful	VERIFY
	GetStatus after failure to verify	Verification failed	VERIFY
	Verify completes successfully	Send VerifyComplete command with successful VerifyResult	APPLY
	Verify ended with failure	Send VerifyComplete command with failed VerifyResult	VERIFY
	CancelUpdateComponent	Success	READY XFER
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	VERIFY
	QueryDownstreamDevices	Success with Downstream Devices	VERIFY
	QueryDownstreamIdentifiers	Success with Downstream Identifiers	VERIFY
	GetFirmwareParameters	Success with firmware info	VERIFY
	GetDownstreamFirmwareParameters	Success with Downstream Device firmware info	VERIFY
	GetMetaData	Success	VERIFY
	FD_T1 timeout waiting for response to VerifyComplete	None	IDLE
	Any other command	Invalid State Machine	VERIFY
APPLY	GetStatus while applying	Apply in progress	APPLY
	GetStatus after successful apply	Apply successful	APPLY
	GetStatus after apply failure	Apply failed	APPLY
	Apply completes successfully	Send ApplyComplete command with successful ApplyResult	READY XFER
	Apply ended with failure	Send ApplyComplete command with failed ApplyResult	APPLY
	CancelUpdateComponent	Success	READY XFER
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	APPLY
	QueryDownstreamDevices	Success with Downstream Devices	APPLY
	QueryDownstreamIdentifiers	Success with Downstream Identifiers	APPLY
	GetFirmwareParameters	Success with firmware info	APPLY

	GetDownstreamFirmwareParameters	Success with Downstream Device firmware info	APPLY
	GetMetaData	Success	APPLY
	FD_T1 timeout waiting for response to ApplyComplete	None	IDLE
	Any other command	Invalid State Machine	APPLY
ACTIVATE	Sets transferred component image to become active firmware component on next activation	Success	IDLE
	Self-contained activation option was requested from READY XFER state for applicable components	Activation is in process	ACTIVATE
	Self-contained activation completes	Idle state	IDLE
	GetStatus	Activate state	ACTIVATE
	QueryDeviceIdentifiers	Success with Identifiers	ACTIVATE
	QueryDownstreamDevices	Success with Downstream Devices	ACTIVATE
	QueryDownstreamIdentifiers	Success with Downstream Identifiers	ACTIVATE
	GetFirmwareParameters	Success with firmware info	ACTIVATE
	GetDownstreamFirmwareParameters	Success with Downstream Device firmware info	ACTIVATE
	GetMetaData	Success	ACTIVATE
	Any other command	Invalid State Machine	ACTIVATE

9.3 State Transition Diagram

The below diagram illustrates the state transitions the FD shall implement. Each bubble represents a particular state as defined in Table 15. Upon initialization, system reboot, or a device reset the FD shall enter the IDLE state. The dashed lines represent state change transitions, not due to timeouts, which are initiated by the FD while the solid lines indicate transitions that are initiated by the UA.

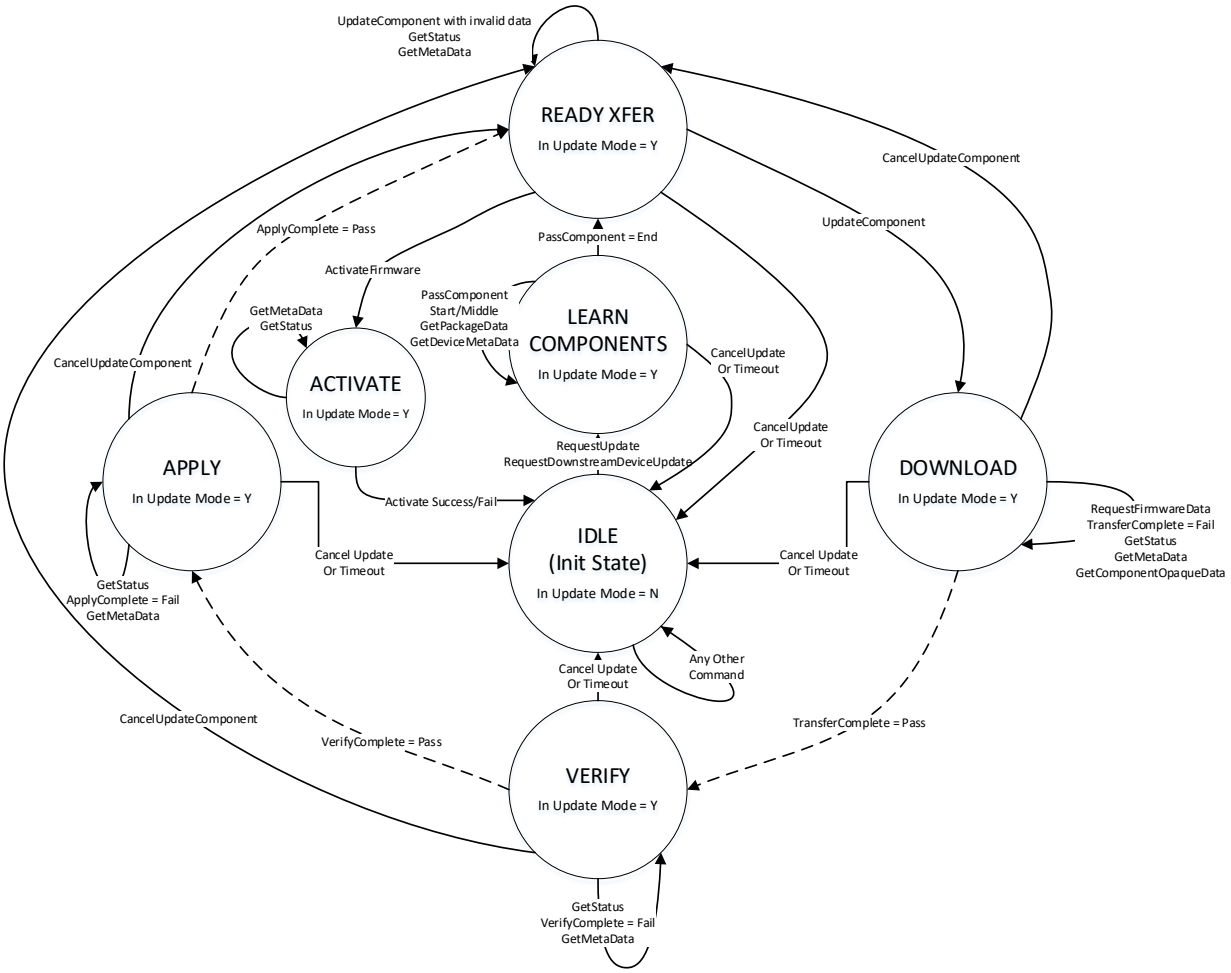


Figure 10 – Firmware Device State Transition Diagram

10 PLDM Commands for Firmware Update

This section provides the list of command codes that are used by Update Agents and Firmware Devices which implement PLDM Firmware Updates as defined in this specification. The command codes for the PLDM messages are given in Table 16.

This specification permits the usage of only a limited number of supported commands for a Firmware Device to provide inventory information only without the ability to update the components. This is known as the 'Inventory Only' function of this specification.

1199 **Table 16 – PLDM for Firmware Update Command Codes**

Command	Command Code	Command Requirement for UA	Command Requirement for FD		Command Requestor (Initiator)	Reference
			FD implementing full update capability	FD implementing inventory only support		
INVENTORY COMMANDS						
QueryDeviceIdentifiers	0x01	Mandatory	Mandatory	Mandatory	UA	See 11.1
GetFirmwareParameters	0x02	Mandatory	Mandatory	Mandatory	UA	See 11.2
QueryDownstreamDevices	0x03	Optional	Optional	Optional	UA	See 11.3
QueryDownstreamIdentifiers	0x04	Optional	Optional	Optional	UA	See 11.4
GetDownstreamFirmwareParameters	0x05	Optional	Optional	Optional	UA	See 11.5
Reserved	0x05-0x0F					
UPDATE COMMANDS						
RequestUpdate	0x10	Mandatory	Mandatory	Optional	UA	See 12.1
GetPackageData	0x11	Mandatory	Optional	Optional	FD	See 12.2
GetDeviceMetaData	0x12	Mandatory	Optional	Optional	UA	See 12.3
PassComponentTable	0x13	Mandatory	Mandatory	Optional	UA	See 12.4
UpdateComponent	0x14	Mandatory	Mandatory	Optional	UA	See 12.5
RequestFirmwareData	0x15	Mandatory	Mandatory	Optional	FD	See 12.6
TransferComplete	0x16	Mandatory	Mandatory	Optional	FD	See 12.7
VerifyComplete	0x17	Mandatory	Mandatory	Optional	FD	See 12.8
ApplyComplete	0x18	Mandatory	Mandatory	Optional	FD	See 12.9
GetMetaData	0x19	Mandatory	Optional	Optional	FD	See 12.10
ActivateFirmware	0x1A	Mandatory	Mandatory	Optional	UA	See 12.11
GetStatus	0x1B	Mandatory	Mandatory	Optional	UA	See 12.12
CancelUpdateComponent	0x1C	Mandatory	Mandatory	Optional	UA	See 12.13
CancelUpdate	0x1D	Mandatory	Mandatory	Optional	UA	See 12.14
ActivatePendingComponentImageSet	0x1E	Optional	Optional	Optional	UA	See 12.15
ActivatePendingComponentImage	0x1F	Optional	Optional	Optional	UA	See 12.16
RequestDownstreamDeviceUpdate	0x20	Optional	Optional	Optional	UA	See 12.17
GetComponentOpaqueData	0x21	Mandatory	Optional	Optional	FD	See 12.18
UpdateSecurityRevision	0x22	Optional	Optional	Optional	UA	See 12.19

11 PLDM for Firmware Update – Inventory Commands

This section describes the commands that are used by Update Agents and Firmware Devices that implement the inventory commands which are defined in this specification. The command codes for the PLDM messages are given in Table 16.

11.1 QueryDeviceIdentifiers Command Format

This command is used by the UA to obtain the firmware identifiers for the FD. The FD shall provide a response message to this command in all states, including IDLE.

Table 17 – QueryDeviceIdentifiers command format

Type	Request data
--	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES }
uint32	**DeviceIdentifiersLength** Contains the length, in bytes, of the Descriptors field.
uint8	**DescriptorCount** The total number of descriptors for the FD.
Variable	**Descriptors** Refer to Table 7 for details on the format and values for these fields.

11.2 GetFirmwareParameters Command Format

The UA sends GetFirmwareParameters command to acquire the component details such as classification types and corresponding versions of the FD. The FD shall provide a response message to this command in all states, including IDLE.

Table 18 – GetFirmwareParameters command format

Type	Request data
--	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES }

bitfield32	**CapabilitiesDuringUpdate** 32 bit field, specifying the capability of the firmware device. Bit [31:10] – Reserved Bit [9] – Security revision number update request support 0: Firmware Device does not support control of component's security revision number update. 1: Firmware Device may have components with a security revision number capability. If this bit is set, the UA may request, via the Security Revision Number Delayed Update bit in the UpdateComponent command, a delay to the update of the security revision for a component image and use the UpdateSecurityRevision command after a firmware update. Refer to the individual component CapabilitiesDuringUpdates field to determine if a specific component supports a security revision number as this bit will provide the general FD capability, but individual components may or may not support the security revision number update request capability. Bit [8] – Firmware device downgrade restrictions 0: Firmware Device does not have downgrade restrictions which may prevent a component image from being downgraded. 1: Firmware Device supports downgrade restrictions, and each component image will report whether a downgrade to an older component image can occur. If this bit is set to 1, then the value of bit [2] in CapabilitiesDuringUpdate of the component image will provide the information for the currently active image. Bit [7:4] – Firmware Device Update Mode Restrictions Bit 4: 0 – No host OS environment restriction for update mode 1 – Firmware device unable to enter update mode if host OS environment is active. Bit 7:5 -- Reserved Bit [3] – Firmware Device Partial Updates 0: Firmware Device cannot accept a partial update and all components present on the FD shall be updated. 1: Firmware Device can support a partial update, whereby a package which contains a component image set that is a subset of all components currently residing on the FD, can be transferred. Note: The UA shall always transfer the entire component image set provided by the firmware update package. No provision is defined within this specification which would allow a UA to only transfer a portion of the component image set. Bit [2] – Firmware Device Host Functionality during Firmware Update 0: Device host functionality is not reduced during Firmware Update. 1: Device host functionality will be reduced, perhaps becoming inaccessible, during Firmware Update. Bit [1] – Component Update Failure Retry Capability 0: Device can have component updated again without exiting update mode and restarting transfer via RequestUpdate command. 1: Device will not be able to update component again unless it exits update mode and the UA sends a new Request Update command. Bit [0] – Component Update Failure Recovery Capability 0: Device will revert to previous component image upon a failure, timeout, or cancelation of the transfer.

	1: Device will not revert to previous component image upon a failure, timeout, or cancelation of the transfer. Therefore the current pending component version may be corrupt if the transfer does not complete.
uint16	**ComponentCount** Number of firmware components which reside within the FD. Each one will have an entry in the following ComponentParameterTable.
enum8	**ActiveComponentImageSetVersionStringType** The type of string used in the ActiveComponentImageSetVersionString field. Refer to Table 33 for values.
uint8	**ActiveComponentImageSetVersionStringLength** The length, in bytes, of the ActiveComponentImageSetVersionString.
enum8	**PendingComponentImageSetVersionStringType** The type of string used in the PendingComponentImageSetVersionString field. This field, and all other pending component image set fields, are valid once the firmware device has received the ActivateFirmware command to prepare the firmware components for activation, but the activation method requires further action to enable the pending images to become the actively running code images. Refer to Table 33 for values. If no pending component image set exists, this value shall be set to '0 – Unknown'.
uint8	**PendingComponentImageSetVersionStringLength** The length, in bytes, of the PendingComponentImageSetVersionString. Refer to PendingComponentImageSetVersionStringType field for additional details. If no pending component image set exists, this value shall be set to 0x0.
Variable	**ActiveComponentImageSetVersionString** Component Image Set version information, up to 255 bytes. Contains a variable type string describing the version of the set of component images which are currently active.
Variable	**PendingComponentImageSetVersionString** Component image set version, which is pending activation, up to 255 bytes. The version reported here should be the one that will become active on the next initialization or activation of the components. The pending component image set version value may be same as the active component image set version. Contains a variable type string describing the pending component image set version. Refer to PendingComponentImageSetVersionStringType field for additional details. If no pending component image set exists, this field is zero bytes in length.
Variable	**ComponentParameterTable** Table of component entries for all of the updateable components which reside on the FD. Refer to Table 19 for details.

1213

1214 **Table 19 – ComponentParameterTable -- Entry Format**

Type	Data
uint16	**ComponentClassification** Vendor specific component classification information. Refer to Table 32 for specific values. Special values: 0x0000, 0xFFFF = reserved.

uint16	**ComponentIdentifier** FD vendor selected unique value to distinguish between component images.
uint8	**ComponentClassificationIndex** Used to distinguish identical components that have the same classification and identifier which can use the same component image but the images are stored in different locations in the FD.
uint32	**ActiveComponentComparisonStamp** Optional Firmware component comparison stamp which is currently active. If the firmware component does not provide a component comparison stamp, this value should be set to 0x00000000.
enum8	**ActiveComponentVersionStringType** The type of strings used in the ActiveComponentVersionString field. Refer to Table 33 for values.
uint8	**ActiveComponentVersionStringLength** The length, in bytes, of the ActiveComponentVersionString.
ASCII[8]	**ActiveComponentReleaseDate** Eight byte field containing the date corresponding to the component version level being reported – Format YYYYMMDD. If the firmware component does not provide a date, this value shall be set to ASCII null characters represented by eight 0x00 bytes.
uint32	**PendingComponentComparisonStamp** Optional firmware component comparison stamp which is pending activation. This field, and all other pending component fields, are valid once the firmware device has received the ActivateFirmware command to prepare the firmware component for activation, but the activation method requires further action to enable the pending image to become the actively running code image. If no pending firmware component exists, this value shall be set to 0x00000000.
enum8	**PendingComponentVersionStringType** The type of strings used in the PendingComponentVersionString field. Refer to PendingComponentComparisonStamp field for additional details. Refer to Table 33 for values. If no pending Firmware Component exists, this value shall be set to ‘0 – Unknown’.
uint8	**PendingComponentVersionStringLength** The length, in bytes, of the PendingComponentVersionString. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this value shall be set to 0x0.
ASCII[8]	**PendingComponentReleaseDate** Eight byte field containing the date corresponding to the component version level being reported – Format YYYYMMDD. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this value shall be set to ASCII null characters represented by eight 0x00 bytes

bitfield16	**ComponentActivationMethods** Provides the capability of the FD for firmware activation. Multiple activation methods can be supported. [15:8] – reserved [7] – Supports ActivatePendingComponentImageSet [6] – Supports ActivatePendingImage [5] - AC power cycle [4] - DC power cycle [3] - System reboot [2] - Medium-specific reset [1] - Self-Contained (can be performed upon transmission of ActivateFirmware command) [0] - Automatic (becomes active as the Apply completes, or as download completes if the FD performs an auto-apply)
bitfield32	**CapabilitiesDuringUpdate** 32 bit field, containing capability of the firmware component. Bit [31:5] – Reserved Bit [4] – Security revision number not at latest level – only valid if Bit 3 is also 1 0: Component security revision number is set to the latest level for the currently active component image. 1: Component security revision number is not set to the latest level for the currently active component image. UA may choose to send the UpdateSecurityRevision command to update the security revision number. Bit [3] – Security revision number update request support 0: Component does not support security revision number update request by UA 1: Component does support security revision number update request by UA Bit [2] – Component downgrade capability 0: Component settings permit a downgrade to older versions 1: Component settings do not allow for a downgrade to an older version component image. Bit[1] – Reserved Bit [0] – Firmware Device apply state functionality. 0: Firmware Device will execute an operation during the APPLY state which will include migrating the new component image to its final non-volatile storage destination. 1: Firmware Device performs an ‘auto-apply’ during transfer phase and apply step will be completed immediately.
Variable	**ActiveComponentVersionString** Firmware component version, which is currently active, up to 255 bytes. Contains a variable type string describing the active component version.
Variable	**PendingComponentVersionString** Firmware component version, which is pending activation, up to 255 bytes. The version reported here should be the one that will become active on the next initialization or activation of the component. The pending component version value may be same as the active component version. Contains a variable type string describing the pending component version. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this field is zero bytes in length.

11.3 QueryDownstreamDevicesCommand Format

This command is used by the UA to obtain information on whether the FDP supports downstream device firmware updates, and how many devices are currently available for update. The FDP shall provide a response message to this command in all states, including IDLE.

Table 20 – QueryDownstreamDevices command format

Type	Request data
--	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES }
enum8	**DownstreamDeviceUpdateSupported** 0 - The FDP does not support firmware updates but may report inventory information on downstream devices. 1 – The FDP supports firmware updates for downstream devices
uint16	**NumberOfDownstreamDevices** Contains the total number of downstream devices presently attached to the FDP
uint16	**MaxNumberOfDownstreamDevices** Contains the maximum number of downstream devices that the FDP supports
Bitfield32	**Capabilities** 32 bit field, containing capability of the FDP for supporting downstream devices Bit [31:3] – Reserved Bit [2] – FDP supports ability to update multiple downstream devices simultaneously Note that all simultaneous downstream devices must be of the same type (where all device descriptors match) 0: No support for simultaneous update 1: FDP supports simultaneous update of multiple downstream devices (UA can request this capability in the PassComponentTable command) Bit [1] – FDP supports downstream devices that can be dynamically removed 0: No dynamically removed downstream devices 1: FDP supports dynamically removed downstream devices Bit [0] – FDP supports downstream devices that can be dynamically attached 0: No dynamically attached downstream devices 1: FDP supports dynamically attached downstream devices

11.4 QueryDownstreamIdentifiers Command Format

This command is used by the UA to obtain the firmware identifiers for the downstream devices supported by the FDP. The entire list of all attached downstream devices is provided by the response to QueryDownstreamIdentifiers command. The FDP shall provide a response message to this command in all states, including IDLE.

1225 **Table 21 – QueryDownstreamIdentifiers command format**

Type	Request data
uint32	**DataTransferHandle** A handle that is used to identify a QueryDownstreamIdentifiers data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart.
enum8	**TransferOperationFlag** The operation flag that indicates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01}
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_TRANSFER_HANDLE, INVALID_TRANSFER_OPERATION_FLAG, DOWNSTREAM_DEVICE_LIST_CHANGED }
uint32	**NextDataTransferHandle** A handle that is used to identify the next portion of the transfer.
enum8	**TransferFlag** The transfer flag that indicates what part of the transfer this response represents. Possible values: {Start=0x01, Middle=0x02, End=0x04, StartAndEnd=0x05}
Variable	**Portion of QueryDownstreamIdentifiers response** Returns a portion of the command response. See Table 22 for details If the FDP has negotiated a PartSize as defined by DSP0240 and its NegotiateTransferParameters command, then the maximum size for this field shall be equal to or less than that negotiated value. Otherwise the FDP can determine the size for this field.

1226 **Table 22 – QueryDownstreamIdentifiers Response Definition**

Type	Response data
uint32	**DownstreamDevicesLength** Contains the length, in bytes, of the DownstreamDevices field.
uint16	**NumberOfDownstreamDevices** Contains the total number of downstream devices presently attached to the FD
Variable	**DownstreamDevices** Refer to Table 23 for details on the format and values for these fields.

1227 The content of the DownstreamDevices field is described in the following table.

1228 **Table 23 – DownstreamDevices Definition**

First Downstream Device	
Type	**Definition**
uint16	**DownstreamDeviceIndex** Used to identify which downstream device this set of descriptors is applicable to. Permitted index range 0x0000 – 0x0FFF = Downstream index number 0x1000 - 0xFFFF = Reserved
uint8	**DownstreamDescriptorCount** The total number of downstream descriptors for this downstream device.

Variable	**DownstreamDescriptors** Refer to Table 7 for details on the format and values for these fields.
Optional Additional Downstream Devices (repeated for each device) For each additional device three fields are provided (Index, Count, Descriptors)	
Type	**Definition**
uint16	**AdditionalDownstreamDeviceIndex** Used to identify which downstream device this set of descriptors is applicable to. Permitted index range 0x0000 – 0x0FFF = Downstream index number 0x1000 - 0xFFFF = Reserved
uint8	**AdditionalDownstreamDescriptorCount** The total number of downstream descriptors for this downstream device.
Variable	**AdditionalDownstreamDescriptors** Refer to Table 7 for details on the format and values for these fields.

Error completion codes handling:

- INVALID_TRANSFER_HANDLE: Returned by the FDP if the transfer handle used in the request is invalid.
- INVALID_TRANSFER_OPERATION_FLAG: Returned by the FDP if the transfer operation flag is invalid.
- DOWNSTREAM_DEVICE_LIST_CHANGED: Returned by the FDP if the transfer operation must end because one or more devices are no longer attached or have been added.

11.5 GetDownstreamFirmwareParameters Command Format

The UA sends GetDownstreamFirmwareParameters command to acquire the component details such as classification types and corresponding versions for the downstream devices supported by the FDP. The FDP shall provide a response message to this command in all states, including IDLE.

Table 24 – GetDownstreamFirmwareParameters command format

Type	**Request data**
uint32	**DataTransferHandle** A handle that is used to identify a GetDownstreamFirmwareParameters data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart.
enum8	**TransferOperationFlag** The operation flag that indicates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01}
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_TRANSFER_HANDLE, INVALID_TRANSFER_OPERATION_FLAG, DOWNSTREAM_DEVICE_LIST_CHANGED }
uint32	**NextDataTransferHandle** A handle that is used to identify the next portion of the transfer.

enum8	**TransferFlag** The transfer flag that indicates what part of the transfer this response represents. Possible values: {Start=0x01, Middle=0x02, End=0x04, StartAndEnd=0x05}
Variable	**Portion of GetDownstreamFirmwareParameters response** Returns a portion of the command response. See Table 25 for details If the FDP has negotiated a PartSize as defined by DSP0240 and its NegotiateTransferParameters command, then the maximum size for this field shall be equal to or less than that negotiated value. Otherwise the FDP can determine the size for this field.

1243 **Table 25 – GetDownstreamFirmwareParameters Response Definition**

Type	Response data
bitfield32	**FDPCapabilitiesDuringUpdate** 32 bit field, specifying the capability of the FDP. Bit [31:10] – Reserved Bit [9] – Security revision number update request support 0: FDP does not support control of component's security revision number update. 1: FDP may have components with security revision number capability (UA can request a delay to the update of the security revision for a downstream component image and use the UpdateSecurityRevision command after a firmware update). Refer to the individual component CapabilitiesDuringUpdates field to determine if a specific downstream device supports a security revision number. Bit [8] – FDP downgrade restrictions 0: FDP does not have downgrade restrictions which may prevent a component image from being downgraded. 1: FDP supports downgrade restrictions, and each component image will report whether a downgrade to an older component image can occur. If this bit is set to 1, then the value of bit [2] in CapabilitiesDuringUpdate of the downstream device component image will provide the information for the currently active image. Bit [7:4] – FDP Update Mode Restrictions Bit 4: 0 – No host OS environment restriction for update mode 1 – Firmware device unable to enter update mode if host OS environment is active. Bit 7:5 -- Reserved Bit [3] – Reserved Bit [2] – Downstream Device Host Functionality during Firmware Update 0: Device host functionality is not reduced during Firmware Update. 1: Device host functionality will be reduced, perhaps becoming inaccessible, during Firmware Update. Bit [1] – Component Update Failure Retry Capability 0: Downstream Device can have component updated again without exiting update mode and restarting transfer via RequestUpdate command. 1: Downstream Device will not be able to update component again unless it exits update mode and the UA sends a new Request Update command. Bit [0] – Downstream Device Component Update Failure Recovery Capability 0: Downstream Device will revert to previous component image upon a failure, timeout, or cancelation of the transfer. 1: Downstream Device will not revert to previous component image upon a failure, timeout, or cancelation of the transfer. Therefore the current pending component version may be corrupt if the transfer does not complete.
uint16	**DownstreamDeviceCount** Number of downstream devices which are supported by the FDP. Each one will have an entry in the following ComponentParameterTable with a different DownstreamDeviceIndex value

Variable	**DownstreamDeviceParameterTable** Table of component entries for all of the downstream devices which are supported by the FDP. Refer to Table 26 for details.

1244

1245 **Table 26 – DownstreamDeviceParameterTable -- Entry Format**

Type	Data
uint16	**DownstreamDeviceIndex** Used to identify which downstream device the component information is applicable to. This value is also used in the UpdateComponent ComponentIdentifier field to identify which downstream device should be updated. Permitted index range 0x0000 – 0x0FFF = Downstream index number 0x1000 - 0xFFFF = Reserved
uint32	**ActiveComponentComparisonStamp** Optional Firmware component comparison stamp which is currently active. If the firmware component does not provide a component comparison stamp, this value should be set to 0x00000000.
enum8	**ActiveComponentVersionStringType** The type of strings used in the ActiveComponentVersionString field. Refer to Table 33 for values.
uint8	**ActiveComponentVersionStringLength** The length, in bytes, of the ActiveComponentVersionString.
ASCII[8]	**ActiveComponentReleaseDate** Eight byte field containing the date corresponding to the component version level being reported – Format YYYYMMDD. If the firmware component does not provide a date, this value shall be set to ASCII null characters represented by eight 0x00 bytes.
uint32	**PendingComponentComparisonStamp** Optional firmware component comparison stamp which is pending activation. This field, and all other pending component fields, are valid once the firmware device has received the ActivateFirmware command to prepare the firmware component for activation, but the activation method requires further action to enable the pending image to become the actively running code image. If no pending firmware component exists, this value shall be set to 0x00000000.
enum8	**PendingComponentVersionStringType** The type of strings used in the PendingComponentVersionString field. Refer to PendingComponentComparisonStamp field for additional details. Refer to Table 33 for values. If no pending Firmware Component exists, this value shall be set to ‘0 – Unknown’.
uint8	**PendingComponentVersionStringLength** The length, in bytes, of the PendingComponentVersionString. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this value shall be set to 0x0.

ASCII[8]	**PendingComponentReleaseDate** Eight byte field containing the date corresponding to the component version level being reported – Format YYYYMMDD. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this value shall be set to ASCII null characters represented by eight 0x00 bytes
bitfield16	**ComponentActivationMethods** Provides the capability of the Downstream Device for firmware activation. Multiple activation methods can be supported. [15:8] – reserved [7] – Reserved [6] – Supports ActivatePendingImage [5] - AC power cycle [4] - DC power cycle [3] - System reboot [2] - Medium-specific reset [1] - Self-Contained (can be performed upon transmission of ActivateFirmware command) [0] - Automatic (becomes active as the Apply completes, or as download completes if the downstream device performs an auto-apply)
bitfield32	**CapabilitiesDuringUpdate** 32 bit field, containing capability of the firmware component. Bit [31:5] – Reserved Bit [4] – Security revision number not at latest level – only valid if Bit 3 is also 1 0: Component security revision number is set to the latest level for the currently active component image. Or component image does not support security revision number. 1: Component security revision number is not set to the latest level for the currently active component image. UA may choose to send the UpdateSecurityRevision command to update the security revision number. Bit [3] – Security revision number update request support 0: Component does not support security revision number update request by UA 1: Component does support security revision number update request by UA Bit [2] – Component downgrade capability 0: Component settings permit a downgrade to older versions 1: Component settings do not allow for a downgrade to an older version component image. Bit [1] – Downstream Device is updateable 0: Downstream Device can provide inventory information only 1: Downstream Device can be updated through the FDP Bit [0] – Downstream Device apply state functionality. 0: Downstream Device will execute an operation during the APPLY state which will include migrating the new component image to its final non-volatile storage destination. 1: Downstream Device performs an ‘auto-apply’ during transfer phase and apply step will be completed immediately.

Variable	**ActiveComponentVersionString** Firmware component version, which is currently active, up to 255 bytes. Contains a variable type string describing the active component version.
Variable	**PendingComponentVersionString** Firmware component version, which is pending activation, up to 255 bytes. The version reported here should be the one that will become active on the next initialization or activation of the component. The pending component version value may be same as the active component version. Contains a variable type string describing the pending component version. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this field is zero bytes in length.

Error completion codes handling:

- INVALID_TRANSFER_HANDLE: Returned by the FDP if the transfer handle used in the request is invalid.
- INVALID_TRANSFER_OPERATION_FLAG: Returned by the FDP if the transfer operation flag is invalid.
- DOWNSTREAM_DEVICE_LIST_CHANGED: Returned by the FDP if the transfer operation must end because one or more devices are no longer attached or have been added.

12 PLDM for Firmware Update – Update Commands

This section describes the commands that are used by Update Agents and Firmware Devices that implement the firmware update capability as defined in this specification. The command numbers for the PLDM messages are given in Table 16.

12.1 RequestUpdate Command Format

This is the first PLDM command to initiate a firmware update for an FD.

The FD shall enter update mode if command response indicates success. While the FD is in update mode, it shall not accept another RequestUpdate or RequestDownstreamDeviceUpdate command. In this case, the FD shall return the ALREADY_IN_UPDATE_MODE completion code.

If the FD is unable to enter update mode to begin a transfer due to other operations or the current operating environment it shall return the UNABLE_TO_INITIATE_UPDATE completion code.

Table 27 – RequestUpdate command format

Type	Request data
uint32	**MaximumTransferSize** Specifies the maximum size, in bytes, of the variable payload allowed to be requested by the FD via the RequestFirmwareData command that is contained within a PLDM message. This value shall be equal to or greater than firmware update baseline transfer size. Refer to Section 7.8 for details on the firmware update baseline transfer size.
uint16	**NumberOfComponents** Specifies the number of components that will be passed to the FD during the update. The FD can use this value to compare against the number of PassComponentTable commands received.

uint8	**MaximumOutstandingTransferRequests** Specifies the number of outstanding RequestFirmwareData commands that can be sent by the FD. The minimum required value is ‘1’ which the UA shall support. It is optional for the UA to support a value higher than ‘1’ for this field.
uint16	**PackageDataLength** This field shall be set to the value contained within the FirmwareDevicePackageDataLength field that was provided in the firmware package header. If no firmware package data was provided in the firmware update package then this length field shall be set to 0x0000.
enum8	**ComponentImageSetVersionStringType** The type of string used in the ComponentImageSetVersionString field. Refer to Table 33 for values.
uint8	**ComponentImageSetVersionStringLength** The length, in bytes, of the ComponentImageSetVersionString.
Variable	**ComponentImageSetVersionString** Component Image Set version information, up to 255 bytes. Contains a variable type string describing the version of the set of component images which will be transferred to the FD.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, ALREADY_IN_UPDATE_MODE, UNABLE_TO_INITIATE_UPDATE, RETRY_REQUEST_UPDATE }
uint16	**FirmwareDeviceMetaDataLength** This field shall be set to the length of the metadata that the FD needs the UA to retain during the firmware update process. If the firmware device has no metadata to be retained during the firmware update process then this length field shall be set to 0x0000.
uint8	**FDWillSendGetPackageDataCommand** Set to 0x02 if the PackageDataLength field indicated that there was package data which the FD should obtain, and the FD will request this data at the beginning of the learn components state, and the FD requires a limit on the amount of bytes transferred by the UA in the response to GetPackageData. This value shall be provided in the GetPackageDataMaximumTransferSize field. Set to 0x01 if the PackageDataLength field indicated that there was package data which the FD should obtain, and the FD will request this data at the beginning of the learn components state. Set to 0x00 if the PackageDataLength field was 0x0000, or if there was package data but the FD does not support the optional GetPackageData command. All other values reserved
uint16	**GetPackageDataMaximumTransferSize** This field is only present if FDWillSendGetPackageDataCommand is set to 0x02. This value defines the maximum length that the UA can send in bytes when responding to a GetPackageData command.

Error completion codes handling:

- ALREADY_IN_UPDATE_MODE: returned by the FD if the device is already in update mode from either a RequestUpdate or RequestDownstreamDeviceUpdate. This may happen when the UA loses connection with the FD in the previous update operation due to an unexpected error. In this case, the UA may send CancelUpdate command requesting the FD to exit from update mode.

- UNABLE_TO_INITIATE_UPDATE: The FD is not able to enter update mode to begin the transfer. The FD shall remain in IDLE state.

- RETRY_REQUEST_UPDATE: The FD is not able to enter update mode immediately. The UA should resend the RequestUpdate command after a delay of UA_T4 as the FD needs more time to prepare to enter update mode. The FD shall remain in IDLE state.

12.2 GetPackageData Command Format

The FD sends this command to transfer optional data that shall be received prior to transferring components during the firmware update process. This command is only used if the firmware update package contained content within the FirmwareDevicePackageData field, the UA provided the length of the package data in the RequestUpdate command, and the FD indicated that it would use this command in the FDWillSendGetPackageDataCommand field.

If the FD indicated that this command will be sent with a 0x01 value in the FDWillSendGetPackageDataCommand field, the UA should not send the GetDeviceMetaData (if applicable) or the PassComponentTable command until the FD completes the entire process of transferring the Package Data from the UA. If there are any errors in the GetPackageData transfer or the FD does not accept the Package Data as valid, it can return the PACKAGE_DATA_ERROR code in the next command received from the UA to report this condition and the UA should cancel the firmware update.

Table 28 – GetPackageData command format

Type	Request data
uint32	**DataTransferHandle** A handle that is used to identify a GetPackageData data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart.
enum8	**TransferOperationFlag** The operation flag that indicates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01}
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED, NO_PACKAGE_DATA, INVALID_TRANSFER_HANDLE, INVALID_TRANSFER_OPERATION_FLAG }
uint32	**NextDataTransferHandle** A handle that is used to identify the next portion of the transfer.
enum8	**TransferFlag** The transfer flag that indicates what part of the transfer this response represents. Possible values: {Start=0x01, Middle=0x02, End=0x04, StartAndEnd=0x05}
Variable	**PortionOfPackageData** A portion of the package data that the UA obtained from the firmware update package. If the FD provided a value in the GetPackageDataMaximumTransferSize field, then the UA should select the amount of data to return such that the byte length for this field, except when TransferFlag = End or StartAndEnd, is equal to or less than that value. If the FD did not provide a value in the GetPackageDataMaximumTransferSize field, the UA should select the amount of data to return such that the byte length for this field, except when TransferFlag = End or StartAndEnd, is equal to or between the values of the firmware update baseline transfer size and MaximumTransferSize from the RequestUpdate or RequestDownstreamDeviceUpdate command. When TransferFlag = End or StartAndEnd, the variable size of this field can also be less than the firmware update baseline transfer size.

Error completion codes handling:

- COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not expected based on the sequence defined to update a firmware component.
- NO_PACKAGE_DATA: Returned by the UA if there is no firmware package data that needs to be sent to the FD.
- INVALID_TRANSFER_HANDLE: Returned by the UA if the transfer handle used in the request is invalid.
- INVALID_TRANSFER_OPERATION_FLAG: Returned by the UA if the transfer operation flag is invalid.

12.3 GetDeviceMetaData Command Format

The UA sends this command to acquire optional data that the FD shall transfer to the UA prior to beginning the transfer of component images. This command is only used if the FD has indicated in the RequestUpdate command response that it has data that shall be retrieved and restored by the UA. The firmware device metadata retrieved by this command will be sent back to the FD through the GetMetaData command after all component images have been transferred.

Table 29 – GetDeviceMetaData command format

Type	Request data
uint32	**DataTransferHandle** A handle that is used to identify a GetDeviceMetaData data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart.
enum8	**TransferOperationFlag** The operation flag that indicates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01}
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_STATE_FOR_COMMAND, NO_DEVICE_METADATA, INVALID_TRANSFER_HANDLE, INVALID_TRANSFER_OPERATION_FLAG, PACKAGE_DATA_ERROR }
uint32	**NextDataTransferHandle** A handle that is used to identify the next portion of the transfer.
enum8	**TransferFlag** The transfer flag that indicates what part of the transfer this response represents. Possible values: {Start=0x01, Middle=0x02, End=0x04, StartAndEnd=0x05}
Variable	**PortionOfMetaData** A portion of the firmware device metadata that the UA shall obtain and retain during the firmware update process. The FD should select the amount of data to return such that the byte length for this field, except when TransferFlag = End or StartAndEnd, is equal to or between the values of the firmware update baseline transfer size and MaximumTransferSize from the RequestUpdate or RequestDownstreamDeviceUpdate command. When TransferFlag = End or StartAndEnd, the variable size of this field can also be less than the firmware update baseline transfer size.

Error completion codes handling:

- INVALID_STATE_FOR_COMMAND: The FD only expects this command in LEARN COMPONENTS state.

- NO_DEVICE_METADATA: Returned by the FD if there is no metadata that needs to be transferred to the UA.

- INVALID_TRANSFER_HANDLE: Returned by the FD if the transfer handle used in the request is invalid.

- INVALID_TRANSFER_OPERATION_FLAG: Returned by the FD if the transfer operation flag is invalid

- PACKAGE_DATA_ERROR: Returned by the FD if the FD previously used the GetPackageData command to obtain package data and determined an error or invalid package data was received. The FD will not continue with the firmware update process and the UA should cancel the update.

12.4 PassComponentTable Command Format

PassComponentTable command is used to pass component information to the FD after the FD enters update mode. The PassComponentTable command contains the component information table for a specific component including ComponentClassificationIndex, ComponentClassification, and version details.

If the firmware update package contains more than one component, multiple PassComponentTable commands are required to be sent by the UA (one for each component). The UA shall pass the component table for all applicable components listed in the firmware package header in ascending order of index.

By receiving the component table, the FD possesses the knowledge of which component(s) are going to be updated. The UA shall set the TransferFlag field to indicate whether the command represents the start, middle, end, or both start and end of the table transfer. Upon receiving the end notification, this indicates to the FD that the entire list has been sent and the FD should transition to the READY XFER state.

Table 30 – PassComponentTable command format

Type	Request data
enum8	**TransferFlag** The transfer flag that indicates what part of the Component Table this request represents. Possible values: {Start = 0x1, Middle = 0x2, End = 0x4, StartAndEnd = 0x5}
uint16	**ComponentClassification** Vendor specific component classification information. Refer to Table 32 for specific values. Special values: 0x0000 If ComponentClassification = 0xFFFF, this indicates the component image is for a downstream device

uint16	**ComponentIdentifier** For a FD component image this field represents the FD vendor selected unique value to distinguish between component images. If the ComponentClassification field = 0xFFFF, then the value in this field shall equal the Downstream Device index number of the downstream device attached to the FDP which the UA is requesting to be updated Values applicable when ComponentClassification Field = 0xFFFF 0x0000 – 0x0FFF = Downstream index number to be updated 0x1000 - 0xFFFF = Reserved
uint8	**ComponentClassificationIndex** For a FD component image this field represents the component classification index which was obtained from the GetFirmwareParameters command to indicate which firmware component the information contained within this command is applicable for. If the ComponentClassification field = 0xFFFF, then this field will be used to identify whether a single downstream device is targeted for the component image update, or multiple downstream devices. Applicable values if ComponentClassification field = 0xFFFF 0x00 = Update only 1 device 0xFF = Update all downstream devices that have exactly the same device descriptors as the specified ComponentIdentifier (the selected Downstream Device index number)
uint32	**ComponentComparisonStamp** FD vendor selected value to use as a comparison value in determining if a firmware component is down-level or up-level. For the same component identifier, the greater of two component comparison stamps is considered up-level compared to the other when performing an unsigned integer comparison.
enum8	**ComponentVersionStringType** The type of strings used in the ComponentVersionString field. Refer to Table 33 for values.
uint8	**ComponentVersionStringLength** The length, in bytes, of the ComponentVersionString.
Variable	**ComponentVersionString** Firmware component version information up to 255 bytes. Contains a variable type string describing the component version.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE, INVALID_STATE_FOR_COMMAND, PACKAGE_DATA_ERROR }
enum8	**ComponentResponse** The FD should reply back with initial compatibility with component provided by UA. 0 – Component can be updated – ComponentResponseCode shall be set to 0x00. 1 – Component may be updateable – A ComponentResponseCode greater than zero shall be provided to explain the reason why the component cannot be updated, or if a flag is required to be set in UpdateOptionFlags field within the UpdateComponent request. All other values reserved.

uint8	**ComponentResponseCode** 0x00: Component can be updated. 0x01: Component comparison stamp is identical to the firmware component comparison stamp in the FD or downstream device. Force update option flag (if supported by FD or FDP) will need to be set in the UpdateComponent request. 0x02: Component comparison stamp is lower than the firmware component comparison stamp in the FD or downstream device. Force update option flag (if supported by FD or FDP) will need to be set to in the UpdateComponent request. 0x03: Invalid component comparison stamp. 0x04: Component has conflict with another component provided in a separate PassComponentTable command. 0x05: Pre-requisites for this component have not been met. 0x06: Component is not supported on FD or Downstream Device 0x07: Security restrictions prevent component from being downgraded. Only applicable when component image is downlevel to currently active component image. 0x08: Incomplete component image set was received. The FD or FDP will reject each UpdateComponent command with response code of 0x08. 0x09: If this new component image is activated, FD or Downstream device will not be able to subsequently update to the currently running active component image. 0x0A: Component version string is identical to the firmware component version string in the FD or downstream device. Force update option flag (if supported by FD or FDP) will need to be set in the UpdateComponent request. This response code can be used only when component comparison stamp is not supported by the FD or FDP. 0x0B: Component version string is lower to the firmware component version string in the FD or downstream device. Force update option flag (if supported by FD or FDP) will need to be set in the UpdateComponent request. This response code can be used only when component comparison stamp is not supported by the FD or FDP. 0x0C–0xCF: Reserved 0xD0–0xEF: Firmware Device or FDP Vendor defined component response code. When an FD or FDP uses a vendor defined status code, it shall also provide its Vendor ID information by using either the PCIe or IANA Vendor descriptor type; a downstream device may also use the IEEE Assigned Company ID or SCSI Vendor ID to provide its Vendor ID information. For details refer to Table 8. 0xF0–0xFF: Reserved

Error completion code handling:

- NOT_IN_UPDATE_MODE: Returned by the FD/FDP if it's not currently in update mode.
- INVALID_STATE_FOR_COMMAND: The FD/FDP only expects this command in LEARN COMPONENTS state.
- PACKAGE_DATA_ERROR: Returned by the FD/FDP if the FD previously used the GetPackageData command to obtain package data and determined an error or invalid package data was received. The FD/FDP will not continue with the firmware update process and the UA should cancel the update.

12.5 UpdateComponent Command Format

The UA sends UpdateComponent command to request updating a specific firmware component.

1356 **Table 31 – UpdateComponent command format**

Type	Request data
uint16	**ComponentClassification** Classification value provided by the firmware package header information for the component to be transferred. Values for this field are aligned with the Value Map from CIM_SoftwareIdentity.Classifications. Refer to Table 32 for values. If ComponentClassification = 0xFFFF, this indicates the component image is for a downstream device and the ComponentIdentifier field will indicate which downstream device is to be updated.
uint16	**ComponentIdentifier** FD Vendor selected unique value to distinguish between component images. If the ComponentClassification field = 0xFFFF, then the value in this field shall equal the Downstream Device index number of the downstream device attached to the FDP which the UA is requesting to be updated Values applicable when ComponentClassification Field = 0xFFFF 0x0000–0x0FFF = Downstream index number to be updated 0x1000–0xFFFF = Reserved
uint8	**ComponentClassificationIndex** The component classification index which was obtained from the GetFirmwareParameters command to indicate which firmware component should be updated. If the ComponentClassification field = 0xFFFF, then this field will be used to identify whether a single downstream device is targeted for the component image update, or multiple downstream devices. Applicable values if ComponentClassification field = 0xFFFF 0x00 = Update only 1 device 0xFF = Update all downstream devices that have exactly the same device descriptors as the specified ComponentIdentifier (the selected Downstream Device index number)
uint32	**ComponentComparisonStamp** FD or downstream device vendor selected value to use as a comparison value in determining if a firmware component is down-level or up-level. For the same component identifier, the greater of two component comparison stamps is considered up-level compared to the other when performing an unsigned integer comparison.
uint32	**ComponentImageSize** Size in bytes of the component image.

bitfield32	**UpdateOptionFlags** 32 bits field, where each non-reserved bit represents an update option that can be requested by the UA to be enabled for the transfer of this component image. [31:3] – reserved [2] – Security Revision Number Delayed Update When set, the UA requests that the FD/FDP updates the component image but does not update the security revision number that is associated with the new component image. This allows for the image to be transferred and activated but the UA can still request a downgrade prior to issuing an UpdateSecurityRevision command. [1] – Component Opaque Data When set, the UA has additional component opaque data which was provided within the firmware update package header. The FD/FDP shall indicate whether this data will be requested with the corresponding UpdateOptionFlagsEnabled bit in the response. [0] – Request Force Update of component – Can be used to inform the FD/FDP to perform a transfer even if the component has a lower or equal component comparison stamp, or version string, than what is currently installed. The UA will set this bit for any component which has the force update bit set in the ComponentOptions field of the package header. Additionally, the UA could set the bit as instructed by commands used to provide the update package to the UA (these commands are out of scope for this spec).
enum8	**ComponentVersionStringType** The type of strings used in the ComponentVersionString field. Refer to Table 33 for values.
uint8	**ComponentVersionStringLength** The length, in bytes, of the ComponentVersionString.
Variable	**ComponentVersionString** Firmware component version information up to 255 bytes. Contains a variable type string describing the component version.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE, INVALID_STATE_FOR_COMMAND}
enum8	**ComponentCompatibilityResponse** The FD/FDP should reply back with initial compatibility with component provided by UA. 0 – Component can be updated, and the FD/FDP will begin to request data via the RequestFirmwareData command. ComponentCompatibilityResponseCode shall be set to 0x00. 1 – Component will not be updated, and the FD/FDP will not begin to request component image data. A ComponentCompatibilityResponseCode greater than zero shall be provided to explain the reason for the FD/FDP rejection of the component. All other values reserved.

uint8	**ComponentCompatibilityResponse Code** 0x00: No response code – used when component can be updated. 0x01: Component comparison stamp is identical to the firmware component comparison stamp in the FD or downstream device, but force update flag is not set. Force update option flag (if supported by FD or FDP) will need to be set to update component. Can also be used if FD or FDP does not support force flag. 0x02: Component comparison stamp is lower than the firmware component comparison stamp in the FD or downstream device, but force update flag is not set. Force update option flag (if supported by FD or FDP) will need to be set to update component. Can also be used if FD or FDP does not support force flag. 0x03: Invalid component comparison stamp or version. 0x04: Component has conflict with another component provided in a separate PassComponentTable command. 0x05: Pre-requisites for this component have not been met. 0x06: Component is not supported on FD or downstream device. 0x07: Security restrictions prevent component from being downgraded. Can be used when force update flag is set, but the firmware component cannot be downgraded. 0x08: Component cannot be updated as an Incomplete Component Image Set was received from the PassComponentTable commands. 0x09: Component information does not match details presented from PassComponentTable commands. 0x0A: Component version string is identical to the firmware component version string in the FD or downstream device, but force update flag is not set. Force update option flag (if supported by FD or FDP) will need to be set to update component. Reason code can be used only when component comparison stamp is not supported by the FD or FDP. 0x0B: Component version string is lower to the firmware component version string in the FD or downstream device, but force update flag is not set. Force update option flag (if supported by FD or FDP) will need to be set to update component. Reason code can be used only when component comparison stamp is not supported by the FD or FDP. 0x0C – 0xCF - Reserved 0xD0-0xEF: Firmware Device Vendor defined component response code. When an FD uses a vendor defined status code, it shall also provide its Vendor ID information by using either the PCIe or IANA Vendor descriptor type; a downstream device may also use the IEEE Assigned Company ID or SCSI Vendor ID to provide its Vendor ID information. For details refer to Table 8. 0xF0 – 0xFF – Reserved
bitfield32	**UpdateOptionFlagsEnabled** 32 bits field, where each non-reserved bit represents an update option that has been enabled by the FD/FDP for the transfer of this component image. This field provides the response from the FD/FDP to the request made by the UA in the UpdateOptionFlag field A ‘1’ in the bit indicates the requested update option flag was accepted. [31:3] – Reserved [2] – Security Revision Number Delayed Update; the FD/FDP will not update the security revision during the firmware transfer or activation process. The UA shall send a UpdateSecurityRevision command to update the security revision number. [1] - Component Opaque Data; the FD/FDP will request opaque data. If this flag is set to 1, the FD/FDP must provide a value in the [0] – Force Update of component; FD/FDP will perform a force update of the component.
uint16	**EstimatedTimeBeforeSendingRequestFirmwareData** Amount of time the FD requires to prepare before sending the first RequestFirmwareData command. Measured in seconds. If this field contains a non-zero value, the UA should not begin any of the timers listed in Table 2 until after the amount of time present in this field has elapsed. It is permissible for the FD to begin sending the RequestFirmwareData commands prior to when the timer would have elapsed.

uint16	**GetComponentOpaqueDataMaximumTransferSize** This field is only present if Bit 1 in UpdateOptionFlagsEnabled is set to 1. This value defines the maximum length that the UA can send in bytes when responding to a GetComponentOpaqueData command.

1357

1358 **Table 32 – ComponentClassification Values**

Value	Package Classification Type
0x0000	**Unknown**
0x0001	**Other**
0x0002	**Driver**
0x0003	**Configuration Software**
0x0004	**Application Software**
0x0005	**Instrumentation**
0x0006	**Firmware/BIOS**
0x0007	**Diagnostic Software**
0x0008	**Operating System**
0x0009	**Middleware**
0x000A	**Firmware**
0x000B	**BIOS/FCode**
0x000C	**Support/Service Pack**
0x000D	**Software Bundle**
0x8000-0xFFFE	**Reserved for Vendor Defined values**
0xFFFF	**Downstream Device**

1359

1360 **Table 33 – String Type Values**

Value	String Type
0	**Unknown**
1	**ASCII**
2	**UTF-8**
3	**UTF-16**
4	**UTF-16LE**
5	**UTF-16BE**

1361

1362 Error completion codes handling:

1363 • NOT_IN_UPDATE_MODE: Returned by the FD/FDP if it’s not currently in update mode.

- INVALID_STATE_FOR_COMMAND: The FD/FDP only expects this command in READY XFER state.

12.6 RequestFirmwareData Command Format

In order for the FD/FDP to retrieve a section of a component image, the FD/FDP sends RequestFirmwareData request message to the UA, specifying its offset and length. The UA will send a response message that includes the component image portion specified by the offset and length from the request message. The FD/FDP shall not request an offset and length values which would extend beyond the end of the component image by more than the firmware update baseline transfer size.

The length of the payload in the response message shall match the length field specified in the request message, otherwise the FD/FDP shall drop the response data and resend the RequestFirmwareData command.

The FD/FDP can request the same data more than one time if it wants to perform an immediate verification of the data. The UA shall allow the FD/FDP to request data at any valid offset within the firmware data. An FDP may also request the same data multiple times if it was requested to update multiple downstream devices of the same type (where all downstream device descriptors match).

Table 34 – RequestFirmwareData command format

Type	Request data
uint32	**Offset** Offset of the component image segment within the current component being transferred.
uint32	**Length** Size of the component image segment requested by the FD/FDP. This value shall be set between the firmware update baseline transfer size, and the MaximumTransferSize value from the RequestUpdate command. Refer to Section 7.8 for details on the firmware update baseline transfer size.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_TRANSFER_LENGTH, COMMAND_NOT_EXPECTED, DATA_OUT_OF_RANGE, RETRY_REQUEST_FW_DATA, CANCEL_PENDING }

Variable	**ComponentImagePortion** The payload contains the portion corresponding to the component image from Offset to (Offset + Length – 1). The UA shall pad with 00s if the length requested extends past the end of the component image. The maximum amount of padding the UA shall support is equal to the firmware update baseline transfer size. Any request from the FD/FDP which would require a larger amount of pad bytes shall have its completion code set to DATA_OUT_OF_RANGE and no data is returned. Refer to Section 7.8 for details on the firmware update baseline transfer size. The permitted range of this ComponentImagePortion can be described by the following two equations: • Firmware Update Baseline Transfer Size <= Length <= MaximumTransferSize If this equation is not satisfied the UA shall return INVALID_TRANSFER_LENGTH • Offset + Length <= ComponentImageSize + Firmware Update Baseline Transfer Size If this equation is not satisfied the UA shall return DATA_OUT_OF_RANGE The maximum amount of pad bytes is equal to the firmware update baseline transfer size and can be described by the following equation: • Pad Bytes = Offset + Length – ComponentImageSize Below is an example of three request/responses each of which are within the permitted range for the ComponentImagePortion. ComponentImageSize = 160 bytes MaximumTransferSize = 512 bytes FD/FDP uses Length = 64 bytes Request #1 Offset = 0, Length = 64 Response #1 UA returns 64 bytes (Offset 0-63) from component image Request #2 Offset = 64, Length = 64 Response #2 UA returns 64 bytes (Offset 64-127) from component image Request #3 Offset = 128, Length = 64 Response #3 UA returns 32 bytes (Offset 128-159) from component image and 32 pad bytes of 0x00

Error completion codes handling:

- INVALID_TRANSFER_LENGTH: The length of the requested component image portion exceeds the MaxTransferSize in the RequestUpdate command, or is less than the firmware update baseline transfer size.

- COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not expected based on the sequence defined to update a firmware component.

- DATA_OUT_OF_RANGE: The requested component image portion offset exceeds the range of the component image, or would require the UA to pad the response with a number of bytes that is larger than the firmware update baseline transfer size. The FD/FDP can send another RequestFirmwareData command to attempt a retry with a different offset and length value.

- RETRY_REQUEST_FW_DATA: The requested component image portion is not currently available from the UA. The UA requests that the firmware device retry this command after FD_T2 as it may be retrieving the component image data from an external source.

- CANCEL_PENDING: The requested component image portion is not returned by the UA as it previously sent a CancelUpdate or CancelUpdateComponent command to the FD/FDP.

12.7 TransferComplete Command Format

The FD/FDP sends TransferComplete command to the UA once the FD/FDP has transferred all the data for the component image or determines the transfer has failed.

If the TransferResult of the request message indicates the transfer completed without error then, upon the successful completion of this command, the FD/FDP proceeds to the next step that verifies the firmware. If the transfer fails, the FD shall remain in the DOWNLOAD state and issue TransferComplete command indicating failed status of the transfer. The UA shall send a CancelUpdateComponent command if a transfer failure occurs

1409 **Table 35 – TransferComplete command format**

Type	Request data
uint8	**TransferResult** Use to indicate the result of the Download stage: 0x00: Transfer has completed without error, no additional information on why is provided with this code. 0x01: Transfer has completed with error as the image received is corrupt 0x02: Transfer has completed with error as the version of the image received does not match the version expected from the UpdateComponent command. 0x03: Firmware Device has aborted the transfer. 0x04 - 0x08: Reserved 0x09: Timeout occurred while performing action. 0x0A: Generic Error has occurred. 0x0B: The FD/FDP has aborted the transfer as the FD/FDP has to enter a low-power state and cannot continue. 0x0C: The FD/FDP has aborted the transfer as it must perform a reset and cannot continue 0x0D: The FD/FDP has aborted the transfer due to an issue with storing the firmware data on the device. 0x0E: The FD/FDP has aborted the transfer due to invalid ComponentOpaqueData which was received, 0x0F: The FD/FDP has aborted the transfer as one or more downstream devices of the same type being updated could not complete the transfer. 0x10: Transfer has completed or aborted with error as the image received will not be updated due to a security revision error. 0x11–0x6F: Reserved 0x70–0x8F: Firmware Device Vendor defined status code. When an FD/FDP uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor descriptor type. For details refer to Table 8. 0x90–0xFF: Reserved When the FD/FDP has a result where multiple choices may be applicable, it should look to provide the most descriptive result code, which is applicable, in this field.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED}

1410 Error completion codes handling:

1411 • COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not
1412 expected based on the sequence defined to update a firmware component.

1413 12.8 VerifyComplete Command Format

1414 After the component image transfer finishes successfully, the FD transitions to the VERIFY state and
1415 performs a validation check against the component image that was received.

1416 The time consumed on verification can be significant depending on the verification algorithm and
1417 hardware performance of the FD controller. The UA may send GetStatus commands to poll the state of
1418 verification from the FD controller.

1419 After the FD finishes verifying the component successfully (including that the image data represents the
1420 expected version that was to be transferred), it issues the VerifyComplete command and transitions to the
1421 APPLY state. If the verification fails, the FD shall remain in the VERIFY state and issue VerifyComplete

command indicating failed status of the verification. The UA shall send a CancelUpdateComponent command if a verification failure occurs

An FDP shall only send the VerifyComplete command after all downstream devices have been verified if it was requested to update multiple downstream devices in the UpdateComponent command.

Table 36 – VerifyComplete command format

Type	Request data
uint8	**VerifyResult** Use to indicate the result of the Verify stage: 0x00: Verify has completed without error. 0x01: Verify has completed with a verification failure – FD will not transition to APPLY state to apply the component. 0x02: Verify has completed with error as the version of the image received does not match the version expected from the UpdateComponent command. – FD will not transition to APPLY state to apply the component. 0x03: Verify has completed with error as the image failed the FD security checks – FD will not transition to the APPLY state to apply the component 0x04: Verify has completed with error as the image transferred was incomplete – FD will not transition to the APPLY state to apply the component 0x05–0x08: Reserved 0x09: Timeout occurred while performing action – FD will not transition to APPLY state to apply the component. 0x0A: Generic Error has occurred – FD will not transition to APPLY state to apply the component. 0x0B–0x0F: Reserved 0x10: Verify has completed with error as the image received will not be updated due to a security revision error. 0x11–0x8F: Reserved 0x90–0xAF: Firmware Device Vendor defined status code. When an FD uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor define type. For details refer to Table 8. 0xB0–0xFF: Reserved When the FD has a result where multiple choices may be applicable, it should look to provide the most descriptive result code, which is applicable, in this field.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED }

Error completion codes handling:

- COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not expected based on the sequence defined to update a firmware component.

12.9 ApplyComplete Command Format

After firmware verification is successful, the FD transitions into the APPLY state and begins transferring the component image into the storage location where the object resides. After the FD finishes applying the component successfully, it issues an ApplyComplete command indicating success and the FD transitions to the READY XFER state to be ready for the next component transfer. If the apply failed, the ApplyComplete command indicates the failure and the FD remains in the APPLY state.

Based on the newly applied component, if the FD determines that the activation method is different than what would be reported in the GetFirmwareParameters or GetDownstreamFirmwareParameters

1438 command prior to the component update, then the FD can set the appropriate bits in the
1439 ComponentActivationMethodsModification field.

1440 An FDP shall only send the ApplyComplete command after all downstream devices have been applied if it
1441 was requested to update multiple downstream devices in the UpdateComponent command.

1442 **Table 37 – ApplyComplete command format**

Type	Request data
uint8	**ApplyResult** Used to indicate the result of the Apply stage: 0x00: Apply has completed without error. 0x01: Apply has completed with success and has modified its activation method. Values shall be provided in the ComponentActivationMethodsModifications field. 0x02: Apply has completed with a failure due to a memory write issue. 0x03 - 0x08: Reserved 0x09: Timeout occurred while performing action. 0x0A: Generic Error has occurred. 0x0B: Apply has completed with error and was not attempted but could be successful if UA re-initiates the transfer. When using this return code, the FD/FDP is requesting the UA to use the CancelUpdate command and restart the transfer as the FD/FDP is aware of a need to transfer the component image(s) a 2nd time. 0x0C – 0x0F: Reserved 0x10: Apply has completed with error as the image received will not be updated due to a security revision error. 0x11 – 0x8F: Reserved 0xB0 – 0xCF: Firmware Device Vendor defined status code. When an FD uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor define type. For details refer to Table 8. 0xD0 – 0xFF: Reserved When the FD has a result where multiple choices may be applicable, it should look to provide the most descriptive result code, which is applicable, in this field.
bitfield16	**ComponentActivationMethodsModification** Field contains a value when the ApplyResult is set to 0x01. Otherwise, each bit shall be set to '0'. Multiple activation methods can be supported. Provides the capability of the FD for firmware activation. This supersedes the values provided by the FD via the GetFirmwareParameters or GetDownstreamFirmwareParameters command. [15:8] – Reserved [7] – Supports ActivatePendingComponentImageSet [6] – Supports ActivatePendingImage [5] - AC power cycle [4] - DC power cycle [3] - System reboot [2] - Medium-specific reset [1] - Self-Contained (can be performed upon transmission of ActivateFirmware command) [0] - Automatic (becomes active as the Apply completes, or as download completes if the FD performs an auto-apply)
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED }

1443 Error completion codes handling:

- COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not expected based on the sequence defined to update a firmware component.

12.10 GetMetaData Command Format

The FD sends this command to transfer the data that was originally obtained by the UA through the GetDeviceMetaData command. This command shall only be used if the FD indicated in the RequestUpdate response that it had device metadata that needed to be obtained by the UA. The FD can send this command when it is in any state, except the IDLE and LEARN COMPONENTS state.

Table 38 – GetMetaData command format

Type	Request data
uint32	**DataTransferHandle** A handle that is used to identify a GetMetaData data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart.
enum8	**TransferOperationFlag** The operation flag that indicates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01}
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED, INVALID_TRANSFER_HANDLE, INVALID_TRANSFER_OPERATION_FLAG }
uint32	**NextDataTransferHandle** A handle that is used to identify the next portion of the transfer.
enum8	**TransferFlag** The transfer flag that indicates what part of the transfer this response represents. Possible values: {Start=0x01, Middle=0x02, End=0x04, StartAndEnd=0x05}
Variable	**PortionOfMetaData** Returns a portion of the metadata that the UA previously obtained from the GetDeviceMetaData command. The UA should select the amount of data to return such that the byte length for this field, except when TransferFlag = End or StartAndEnd, is equal to or between the values of the firmware update baseline transfer size and MaximumTransferSize from the RequestUpdate or RequestDownstreamDeviceUpdate command. When TransferFlag = End or StartAndEnd, the variable size of this field can also be less than the firmware update baseline transfer size.

Error completion codes handling:

- COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not expected based on the sequence defined to update a firmware component, or if the UA did not previously retrieve the firmware device metadata through the GetDeviceMetaData command.

- INVALID_TRANSFER_HANDLE: Returned by the UA if the transfer handle used in the request is invalid.

- INVALID_TRANSFER_OPERATION_FLAG: Returned by the UA if the transfer operation flag is invalid.

12.11 ActivateFirmware Command Format

After all firmware components in the FD have been transferred and applied, the UA sends this command to inform the FD to prepare all successfully applied components to become active at the next activation.

The UA can also request activation of all components that have an activation method of 'Self-Contained'.

The FD shall exit from update mode upon the successful completion of this command, but will first transition to the ACTIVATE state if a self-contained activation is requested and permitted. The FD may not be able to respond to UA commands while in the ACTIVATE state, and will automatically transition to the IDLE state at the conclusion of the self-contained activation. If the command completed with an error code returned, refer to the details for the error code to determine if the FD will transition to IDLE or remain in the READY_XFER state.

The EstimatedTimeForSelfContainedActivation in the response message indicates the maximum time in seconds to finish activation if self-contained activation is requested. The FD controller may not be able to respond to commands when activating firmware. The UA periodically sends "GetStatus" to the FD controller within the maximum activation time to detect if the activation completes.

Table 39 – ActivateFirmware command format

Type	Request data
bool8	**SelfContainedActivationRequest** True: FD/FDP shall activate all self-contained activation capable components. False: FD/FDP shall not activate any self-contained activation capable components. If there are no component images capable of self-contained activation, this field must be set to False.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE, INVALID_STATE_FOR_COMMAND, INCOMPLETE_UPDATE, ACTIVATION_NOT_REQUIRED, SELF_CONTAINED_ACTIVATION_NOT_PERMITTED }
uint16	**EstimatedTimeForSelfContainedActivation** Amount of time the FD requires to perform a self-contained activation. Measured in seconds after sending this response, the UA should not begin any of the timers listed in Table 2 until after the amount of time present in this field has elapsed. If Self-Contained activation is not requested, this field should be set to zero.

Error completion codes handling:

- INCOMPLETE_UPDATE: Returned by the FD/FDP if it is able to determine that not all components are updated completely. The FD/FDP will remain in the READY XFER state, and will not perform activation.
- INVALID_STATE_FOR_COMMAND: The FD/FDP only expects this command in READY XFER state.
- NOT_IN_UPDATE_MODE: Returned by the FD/FDP if it's not in the update mode.
- ACTIVATION_NOT_REQUIRED: Returned by the FD/FDP if the new firmware components are already pending activation (such as through a previous ActivateFirmware command), or the activation method was 'automatic' and therefore the component was already activated at the completion of the apply step. The FD/FDP will transition to the IDLE state and exit update mode as no further action is required by the UA.

- SELF_CONTAINED_ACTIVATION_NOT_PERMITTED: Returned by the FD/FDP if it does not support Self-Contained activation and the SelfContainedActivationRequest is set to True. The FD/FDP will remain in the READY XFER state, and will not perform activation.

12.12 GetStatus Command Format

The UA sends this command to acquire the status of the FD/FDP.

Table 40 – GetStatus command format

Type	Request data
--	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES }
enum8	**CurrentState** Current state machine state of the FD/FDP. 0 – IDLE 1 – LEARN COMPONENTS 2 – READY XFER 3 – DOWNLOAD 4 – VERIFY 5 – APPLY 6 – ACTIVATE
enum8	**PreviousState** The previous different state machine state of the FD/FDP. If the FD/FDP has just been initialized, the PreviousState and CurrentState may both be set to '0 – IDLE' or if the FD/FDP has no ability to recall the last state machine state (if any). 0 – IDLE 1 – LEARN COMPONENTS 2 – READY XFER 3 – DOWNLOAD 4 – VERIFY 5 – APPLY 6 – ACTIVATE
enum8	**AuxState** Used provide additional information to the UA to describe the current operation state of the FD/FDP while in one of the following states (Download, Verify, Apply, Activate, or IDLE). 0 – Operation in progress. 1 – Operation successful. 2 – Operation failed – FD/FDP shall provide Error Code in AuxStateStatus field. 3 – Value used when FD/FDP is in IDLE (except if error after self-contained activation), Learn Components, or Ready Xfer state. 4 – Value used when FD/FDP is in IDLE after a self-contained activation failure.

uint8	**AuxStateStatus** 0x00: AuxState is In Progress or Success. 0x01–0x08: Reserved 0x09: Timeout occurred while performing action. 0x0A: Generic Error has occurred. 0x0B – Self-contained activation failure has occurred. UA may need to retry firmware update process. 0x0C–0x6F: Reserved 0x70–0xEF: Firmware Device Vendor defined status code. When an FD/FDP uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor define type; a downstream device may also use the IEEE Assigned Company ID or SCSI Vendor ID to provide its Vendor ID information. For details refer to Table 7. 0xF0–0xFF: Reserved
uint8	**ProgressPercent** Used when CurrentState is in the DOWNLOAD, VERIFY or APPLY state. Value range from 0x00 to 0x64 (decimal 0 to 100). This field is optional for an FD. If the FD/FDP does not support a progress percent, the value returned shall be 0x65 (decimal 101). If the FD/FDP is expected to take more than 180 seconds in the download, verify, or apply state it should use this field to report status progress to the UA. If this field is supported by the FD/FDP, the value provided in this field represents the percentage complete of the current action (DOWNLOAD, VERIFY, or APPLY). The value is initialized to 0 upon each transition of CurrentState.
enum8	**ReasonCode** Used when CurrentState is in the IDLE state. Provides the reason for why the CurrentState entered the IDLE state. The value is retained until the next transition to IDLE occurs which will then cause this field to be updated. 0 – Initialization of firmware device has occurred. 1 – ActivateFirmware command was received. 2 – CancelUpdate command was received. 3 – Timeout occurred when in LEARN COMPONENT state. 4 – Timeout occurred when in READY XFER state. 5 – Timeout occurred when in DOWNLOAD state. 6 – Timeout occurred when in VERIFY state. 7 – Timeout occurred when in APPLY state. 200–255: Firmware Device Vendor defined status code. When an FD/FDP uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor define type; a downstream device may also use the IEEE Assigned Company ID or SCSI Vendor ID to provide its Vendor ID information. For details refer to Table 8.
bitfield32	**UpdateOptionFlagsEnabled** 32 bits field used when CurrentState is in the DOWNLOAD, VERIFY, APPLY, or ACTIVATE state, where each non-reserved bit represents an update option that has been enabled by the FD/FDP for the transfer of this component image. A ‘1’ in the bit indicates the requested update option flag is enabled. [31:1] – Reserved [0] – Force update of component – FD/FDP will perform a force update of the component.

GetStatus is provided to poll the status of the FD/FDP controller. The timeout waiting for ProgressPercent change is defined by UA_T3. When the UA does not see a change in the ProgressPercent after waiting for UA_T3 time, then the UA can send CancelUpdateComponent command to cancel the component update. If the FD/FDP does not support a ProgressPercent value, the UA will use the timeout defined by UA_T6 to send the CancelUpdateComponent command.

12.13 CancelUpdateComponent Command Format

During the firmware component transfer process, the UA may send this command to the FD/FDP. The FD/FDP, upon receiving this command shall stop sending RequestFirmwareData commands to the UA, and cancel the current component update procedure. The FD/FDP controller shall transition to the READY XFER state of update mode and be ready to accept another UpdateComponent command. The UA may attempt to resend the same component image to the UA.

It is strongly recommended that the entire firmware update procedure be performed as a single sequence of events and not cancelled by the UA. This specification does not describe or provide guidance on a recovery procedure if the FD or downstream device operation is affected by a partially transferred image. After canceling the update, the FD or downstream device may not be able to operate normally if only a portion of the firmware update has been completed.

Table 41 – CancelUpdateComponent command format

Type	Request data
—	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE, BUSY_IN_BACKGROUND, INVALID_STATE_FOR_COMMAND }

Error completion codes handling:

- NOT_IN_UPDATE_MODE: returned by the FD/FDP if it’s not currently in update mode.
- BUSY_IN_BACKGROUND: returned by the FD/FDP if there is a critical job in the background, and cannot exit from update mode. The UA shall retry after UA_T1.
- INVALID_STATE_FOR_COMMAND: The FD/FDP only expects this command in DOWNLOAD, VERIFY, APPLY state.

12.14 CancelUpdate Command Format

This command signals to the FD/FDP that it should exit from update mode even if activation is required to begin operating at the new firmware level. The UA should always attempt to complete the transfer of all components and use this command only if it determines that there is no other method to continue with the transfer process. The FD/FDP will provide a response field which indicates which components will be in a non-functioning state upon exit of update mode and subsequent external activation, such as an initialization of the FD or downstream device. This will depend on the FD's or downstream device’s capability to recover from failed component updates. The indication will allow the UA to understand when a failed FD or downstream device update results in a non-functioning component state which may require recovery actions (outside the scope of this specification) to place the component into a functioning state.

It is strongly recommended that the entire firmware update procedure be performed as a single sequence of events and not cancelled by the UA. This specification does not describe or provide guidance on a recovery procedure if the FD or downstream device operation is affected by a partially transferred image. After canceling the update, the FD or downstream device may not be able to operate normally if only a portion of the firmware update has been completed.

1539 **Table 42 – CancelUpdate command format**

Type	Request data
—	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE, BUSY_IN_BACKGROUND }
bool8	**NonFunctioningComponentIndication** True: one or more components will be in a non-functioning state upon the next activation. The non-functioning component bitmap field indicates which components will be non-functioning. False: all components will be functioning. GetFirmwareParameters can be used to determine the individual component version information. When a UA sends this command to an FDP to cancel an update that began with the RequestDownstreamDeviceUpdate command, then the FDP shall set this field to False even if some downstream devices may be in a non-functioning state. Recovery of downstream devices that may be in a non-functioning state due to the UA sending CancelUpdate is outside the scope of this specification.
bitfield64	**NonFunctioningComponentBitmap** This field is valid only if the Non-functioning component indication field is set to True. Each bit n corresponds to the nth component passed in the PassComponentTable command. A set bit indicates the component will be in a non-functioning state upon the next activation.

1540 Error completion codes handling:

1541 • NOT_IN_UPDATE_MODE: returned by the FD/FDP if it's not in the update mode.
1542
1543 • BUSY_IN_BACKGROUND: returned by the FD/FDP if there are critical tasks already being
1544 performed by the device, and cannot exit from update mode. The UA shall retry within UA_T1
1545 interval.

1546 12.15 ActivatePendingComponentImageSet Command Format

1547 This command can be used to activate the pending component image set of an FD. This command shall
1548 only be sent to an FD that is in the IDLE state, and all component images within the component image set
1549 must support self-contained activation.

1550 The EstimatedTimeForActivation in the response message indicates the maximum time in seconds to
1551 finish activation. The FD controller may not be able to respond to commands when activating firmware.
1552 The UA may periodically send "GetStatus" to the FD controller within the maximum activation time to
1553 detect if the activation completes.

1554 **Table 43 – ActivatePendingComponentImageSet command format**

Type	Request data
--	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_STATE_FOR_COMMAND, ACTIVATION_NOT_REQUIRED, ACTIVATE_PENDING_IMAGE_NOT_PERMITTED }

uint16	**EstimatedTimeForActivation** Amount of time the FD requires to perform a self-contained activation. Measured in seconds after sending this response, the UA should not begin any of the timers listed in Table 2 until after the amount of time present in this field has elapsed.

Error completion codes handling:

- INVALID_STATE_FOR_COMMAND: The FD only expects this command in the IDLE state.
- ACTIVATION_NOT_REQUIRED: The FD does not have a pending component image set that can be activated
- ACTIVATE_PENDING_IMAGE_NOT_PERMITTED: Returned by the FD if it does not support activation of the pending component image set.

12.16 ActivatePendingComponentImage Command Format

This command can be used to activate a pending component image on an FD or a downstream device. This command shall only be sent to an FD or FDP that is in the IDLE state, and the requested component image must support self-contained activation.

The EstimatedTimeForActivation in the response message indicates the maximum time in seconds to finish activation. The FD/FDP controller may not be able to respond to commands when activating firmware. The UA may periodically send “GetStatus” to the FD/FDP controller within the maximum activation time to detect if the activation completes.

1571 **Table 44 – ActivatePendingComponentImage command format**

Type	Request data
uint16	**ComponentClassification** Vendor specific component classification information. Refer to Table 32 for specific values. If ComponentClassification = 0xFFFF, this indicates the component image is for a downstream device and the ComponentIdentifier field will indicate which downstream device is targeted for activation of the pending component image.
uint16	**ComponentIdentifier** FD vendor selected unique value to distinguish between component images. If the ComponentClassification field = 0xFFFF, then the value in this field shall equal the Downstream Device Index number of the downstream device attached to the FDP which the UA is requesting to be activated Values applicable when ComponentClassification Field = 0xFFFF 0x0000 – 0x0FFF = Downstream index number to be activated 0x1000 - 0xFFFF = Reserved
uint8	**ComponentClassificationIndex** Used to distinguish identical components that have the same classification and identifier which can use the same component image but the images are stored in different locations in the FD. If the ComponentClassification field = 0xFFFF, then this field will be used to identify whether a single downstream device is targeted for the component image activation, or multiple downstream devices. Applicable values if ComponentClassification field = 0xFFFF 0x00 = Activate Component Image for only 1 device 0xFF = Activate Component Images for all downstream devices that have exactly the same device descriptors as the specified ComponentIdentifier (the selected Downstream Device index number)
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_STATE_FOR_COMMAND, ACTIVATION_NOT_REQUIRED, ACTIVATE_PENDING_IMAGE_NOT_PERMITTED }
uint16	**EstimatedTimeForActivation** Amount of time the FD requires to perform a self-contained activation. Measured in seconds after sending this command, the UA should not begin any of the timers listed in Table 2 until after the amount of time present in this field has elapsed. If multiple downstream devices have been selected for activation, then this field should provide the total amount of time for all component images across the downstream devices to be activated.

1572 Error completion codes handling:

1573 • INVALID_STATE_FOR_COMMAND: The FD only expects this command in the IDLE state.
1574
1575 • ACTIVATION_NOT_REQUIRED: The requested component identifier and index does not have a
1576 pending image that can be activated
1577
1578 • ACTIVATE_PENDING_IMAGE_NOT_PERMITTED: Returned by the FD/FDP if it does not
1579 support activation of the pending component image.

12.17 RequestDownstreamDeviceUpdate Command Format

This is the first PLDM command to initiate a firmware update for a downstream device. The UA may send this command to an FDP which will act as a proxy for the downstream device that it supports for firmware update using this specification.

The FDP shall enter update mode if command response indicates success. While the FDP is in update mode, it shall not accept another RequestUpdate or RequestDownsteamDeviceUpdate command. In this case, the FDP shall return the ALREADY_IN_UPDATE_MODE completion code.

If the FDP is unable to enter update mode to begin a transfer due to other operations or the current operating environment it shall return the UNABLE_TO_INITIATE_UPDATE completion code.

Table 45 – RequestDownstreamDeviceUpdate command format

Type	Request data
uint32	**MaximumDownstreamDeviceTransferSize** Specifies the maximum size, in bytes, of the variable payload allowed to be requested by the FDP, which will act as the proxy for the Downstream Device during the update, via the RequestFirmwareData command that is contained within a PLDM message. This value shall be equal to or greater than firmware update baseline transfer size. Refer to Section 7.8 for details on the firmware update baseline transfer size.
uint8	**MaximumOutstandingTransferRequests** Specifies the number of outstanding RequestFirmwareData commands that can be sent by the FDP which will act as the proxy for the Downstream Device. The minimum required value is ‘1’ which the UA shall support. It is optional for the UA to support a value higher than ‘1’ for this field.
uint16	**DownstreamDevicePackageDataLength** This field shall be set to the value contained within the DownstreamDevicePackageDataLength field that was provided in the firmware package header. If no Downstream Device package data was provided in the firmware update package then this length field shall be set to 0x0000.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, ALREADY_IN_UPDATE_MODE, UNABLE_TO_INITIATE_UPDATE, RETRY_REQUEST_UPDATE }
uint16	**DownstreamDeviceMetaDataLength** This field shall be set to the length of the metadata that the FDP needs the UA to retain during the firmware update process. If the downstream device has no metadata to be retained during the firmware update process then this length field shall be set to 0x0000.
uint8	**DDWillSendGetPackageDataCommand** Set to 0x02 if the PackageDataLength field indicated that there was package data which the FDP should obtain, and the FDP will request this data at the beginning of the learn components state, and the FDP requires a limit on the amount of bytes transferred by the UA in the response to GetPackageData. This value shall be provided in the GetPackageDataMaximumTransferSize field. Set to 0x01 if the PackageDataLength field indicated that there was package data which the FDP should obtain, and the FDP will request this data at the beginning of the learn components state. Set to 0x00 if the PackageDataLength field was 0x0000, or if there was package data but the FDP does not support the optional GetPackageData command. All other values reserved
uint16	**GetPackageDataMaximumTransferSize** This field is only present if DDWillSendGetPackageDataCommand is set to 0x02. This value defines the maximum length that the UA can send in bytes when responding to a GetPackageData command.

Error completion codes handling:

- ALREADY_IN_UPDATE_MODE: returned by the FDP if the device is already in update mode from either a RequestUpdate or RequestDownstreamDeviceUpdate. This may happen when the UA loses connection with the FDP in the previous update operation due to an unexpected error. In this case, the UA may send CancelUpdate command requesting the FD to exit from update mode.

- UNABLE_TO_INITIATE_UPDATE: The FDP is not able to enter update mode to begin the transfer. The FD shall remain in IDLE state.

- RETRY_REQUEST_UPDATE: The FDP is not able to enter update mode immediately. The UA should resend the RequestDownstreamDeviceUpdate command after a delay of UA_T4 as the FD needs more time to prepare to enter update mode. The FDP shall remain in IDLE state.

12.18 GetComponentOpaqueData Command Format

The FD/FDP sends this command to transfer optional component opaque data during the DOWNLOAD portion of the firmware update process. This command is only used if the firmware update package contained content within the ComponentOpaqueData field, the UA indicated to the FD/FDP in the UpdateComponent command that this data was available, and that the FD/FDP indicated that it would use this command in the UpdateOptionFlagsEnabled Component Opaque Data bit response.

If the FD/FDP indicated that this command will be sent, the FD/FDP can send this command during the DOWNLOAD state. This can occur before or after the RequestFirmwareData command as the FD/FDP may need to obtain this opaque data in a certain sequence with the component image transfer.

If there are any errors in the GetComponentOpaqueData transfer or the FD/FDP does not accept the component opaque data as valid, it can end the DOWNLOAD portion of the transfer by sending the TransferComplete command with an error code to report this condition and the UA should cancel the firmware update.

Table 46 – GetComponentOpaqueData command format

Type	Request data
uint32	**DataTransferHandle** A handle that is used to identify a GetComponentOpaqueData data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart.
enum8	**TransferOperationFlag** The operation flag that indicates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01}
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED, NO_OPAQUE_DATA, INVALID_TRANSFER_HANDLE, INVALID_TRANSFER_OPERATION_FLAG }
uint32	**NextDataTransferHandle** A handle that is used to identify the next portion of the transfer.
enum8	**TransferFlag** The transfer flag that indicates what part of the transfer this response represents. Possible values: {Start=0x01, Middle=0x02, End=0x04, StartAndEnd=0x05}

Variable	**PortionOfComponentOpaqueData** A portion of the component opaque data that the UA obtained from the firmware update package. If the FD provided a value in the GetComponentOpaqueDataMaximumTransferSize field, then the UA should select the amount of data to return such that the byte length for this field, except when TransferFlag = End or StartAndEnd, is equal to or less than that value. If the FD did not provide a value in the GetComponentOpaqueDataMaximumTransferSize field, the UA should select the amount of data to return such that the byte length for this field, except when TransferFlag = End or StartAndEnd, is equal to or between the values of the firmware update baseline transfer size and MaximumTransferSize from the RequestUpdate or RequestDownstreamDeviceUpdate command. When TransferFlag = End or StartAndEnd, the variable size of this field can also be less than the firmware update baseline transfer size.

Error completion codes handling:

- COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not expected based on the sequence defined to update a firmware component.
- NO_OPAQUE_DATA: Returned by the UA if there is no component opaque data that needs to be sent to the FD.
- INVALID_TRANSFER_HANDLE: Returned by the UA if the transfer handle used in the request is invalid.
- INVALID_TRANSFER_OPERATION_FLAG: Returned by the UA if the transfer operation flag is invalid.

12.19 UpdateSecurityRevision Command Format

This command can be used to change the security revision number on a component image if the FD/FDP reported that it can support this feature. This command shall only be sent to an FD or FDP that is in the IDLE state, and the requested component image must support the security revision delayed update capability.

An FD/FDP that supports security revision numbers on components can never downgrade to a component image version that has a lower security revision. Even if the UA requests a force update of the component, the FD/FDP cannot downgrade to a component image with a lower security revision number.

The security revision is a value that is the minimum level to which a new component image transfer must also be at or higher. Typically, this security revision number is updated automatically when needed during the firmware update process. However, the UA can request that this update be delayed and not set during the PLDM firmware transfer and sends this command to update the security revision number. This therefore could be used by the UA to allow for some period of time where the newly transferred image is tested for feature/function support and could still allow a downgrade if needed since the security revision number was not yet updated. The UA cannot request an update to a specific security revision.

This command operates only on the active running component image and not the pending component image.

1648 **Table 47 – UpdateSecurityRevision command format**

Type	Request data
uint16	**ComponentClassification** Vendor specific component classification information. Refer to Table 32 for specific values. If ComponentClassification = 0xFFFF, this indicates the component image is for a downstream device and the ComponentIdentifier field will indicate which downstream device is targeted for security revision update of the pending component image.
uint16	**ComponentIdentifier** FD vendor selected unique value to distinguish between component images. If the ComponentClassification field = 0xFFFF, then the value in this field shall equal the Downstream Device Index number of the downstream device attached to the FDP which the UA is requesting a security revision update Values applicable when ComponentClassification Field = 0xFFFF 0x0000 – 0x0FFF = Downstream index number to have its security revision updated 0x1000 - 0xFFFF = Reserved
uint8	**ComponentClassificationIndex** Used to distinguish identical components that have the same classification and identifier which can use the same component image but the images are stored in different locations in the FD. If the ComponentClassification field = 0xFFFF, then this field will be used to identify whether a single downstream device is targeted for the component image security revision update, or multiple downstream devices. Applicable values if ComponentClassification field = 0xFFFF 0x00 = Update Security Revision of Component Image for only 1 device 0xFF = Update Security Revision of Component Images for all downstream devices that have exactly the same device descriptors as the specified ComponentIdentifier (the selected Downstream Device index number)
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_STATE_FOR_COMMAND, UPDATE_SECURITY_REVISION_NOT_PERMITTED }

1649 Error completion codes handling:

1650 • INVALID_STATE_FOR_COMMAND: The FD only expects this command in the IDLE state.
1651
1652 • UPDATE_SECURITY_REVISION_NOT_PERMITTED: Returned by the FD/FDP if it does not
1653 support updating the security revision number of the component image

13 Additional Information

13.1 Multipart Transfers

The commands GetPackageData, GetDeviceMetaData, GetMetaData, QueryDownstreamIdentifiers, and GetDownstreamFirmwareParameters which are defined in Section 11and 12 for transferring package data, firmware device metadata or downstream device information, support multipart transfers. These commands use flags and data transfer handles to perform multipart transfers. A data transfer handle uniquely identifies the next part of the transfer. The data transfer handle values are implementation specific. For example, an implementation can use memory addresses or sequence numbers as data transfer handles. Following are some requirements for using TransferOperationFlag, TransferFlag, and DataTransferHandle for a given data transfer:

- For initiating a data transfer (or getting the first part of data) using a Get command, the TransferOperationFlag shall be set to GetFirstPart in the request of the Get command.

- For transferring a part other than the first part of data by using a Get command, the TransferOperationFlag shall be set to GetNextPart and the DataTransferHandle shall be set to the NextDataTransferHandle that was obtained in the response of the previous Get command for this data transfer.

- The TransferFlag specified in the response of a Get command has the following meanings:

  – Start, which is the first part of the data transfer

  – Middle, which is neither the first nor the last part of the data transfer

  – End, which is the last part of the data transfer

  – StartAndEnd, which is the first and the last part of the data transfer

- The requester shall consider a data transfer complete when the TransferFlag in the response of a Get command is set to End or StartAndEnd.

The following example shows how the multipart transfers can be performed using the generic mechanism defined in the commands.

In this example, the update agent maintains a copy of the package data provided by the firmware update package. The firmware device gets the package data by using the GetPackageData command. Figure 11 shows the flow of the data transfer.

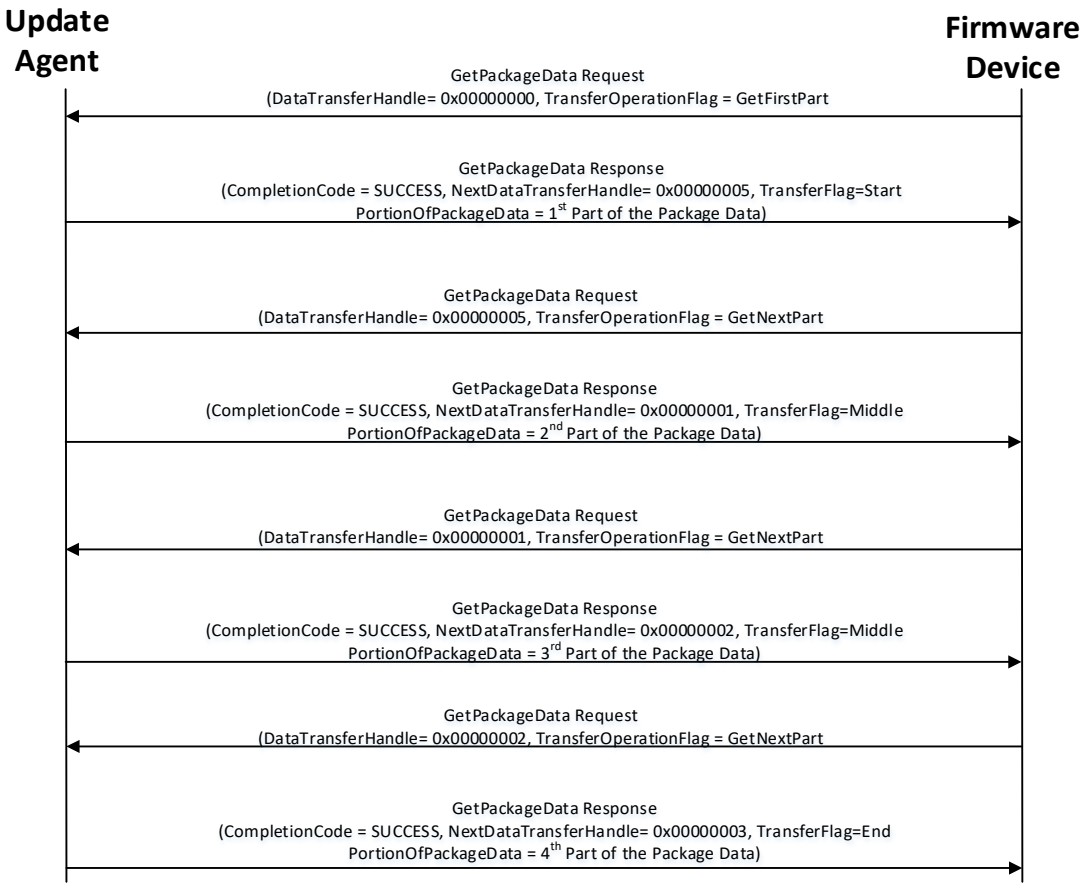


Figure 11 – Multipart Package Data Transfer Using the GetPackageData command

13.2 Transport Protocol Type Supported

PLDM can support bindings over multiple interfaces, refer to DSP0245 for the complete list. This specification requires the transport protocol type to support asynchronous request/response messages which can be sent from either endpoint in order to support the full Firmware Update functionality. All transport protocol types can be supported for the two Inventory commands defined in Table 16.

13.3 Considerations for FD Manufacturers

This specification does not provide a direct recovery method for when the update process is interrupted by power loss, interface failures, or unplanned reboots. An FD manufacturer can look to minimize the exposure to these types of events by implementing a dual bank approach for firmware components. By using a dual bank approach, the new component data being updated is placed into a ‘backup’ image location and the FD would continue to use the actively running image location until an ActivateFirmware command has been received. At that point the FD will enable the new image to become the active running image at the next activation. If a power loss or interruption occurred prior to receiving the ActivateFirmware command the FD would continue to use actively running image and the UA can subsequently restart the firmware update process to update all components again.

ANNEX A
(**informative**)

Change Log

Version	Date	Author	Description
1.0.0	2016-11-28	P. Caporale	DMTF Standard
1.0.1	2018-01-18	P. Caporale	Updates to UUID field in header, PCI descriptors, and activation state machine transition table
1.1.0	2019-07-22	P. Caporale	Add support for Downstream Devices.
1.2.0	2022-09-22	P. Caporale	Add additional support for Opaque data, security features and additional state machine transitions
1.3.0	2023-12-13	P. Caporale	Add support for manifest data, package payload checksum, individual firmware device package, PCI ID revision descriptor range.

Bibliography

DMTF DSP4014, *DMTF Process for Working Bodies 2.6*,
https://www.dmtf.org/sites/default/files/standards/documents/DSP4014_2.6.pdf