



1

Document Identifier: DSP0242

2

Date: 2024-07-29

3

Version: 1.0.0

4

# Platform Level Data Model (PLDM) for File Transfer Specification

5

Supersedes: None

6

Document Class: Normative

7

Document Status: Published

8

Document Language: en-US

Copyright Notice

Copyright © 2024 DMTF. All rights reserved.

- 9 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.
- 10 Implementation of certain elements of this standard or proposed standard may be subject to third-party patent rights, including provisional patent rights (herein “patent rights”). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third-party patent right owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners, or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third-party patent rights, or for such party’s reliance on the standard or incorporation thereof in its product, protocols, or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.
- 11 For information about patents held by third parties which have notified DMTF that, in their opinion, such patents may relate to or impact implementations of DMTF standards, visit <https://www.dmtf.org/about/policies/disclosures>.
- 12 This document’s normative language is English. Translation into other languages is permitted.

1 Foreword	5
1.1 Acknowledgments	5
2 Introduction	6
2.1 Document conventions	6
3 Scope	7
4 Normative references	8
5 Terms and definitions	9
6 Symbols and abbreviated terms	10
7 PLDM for File Transfer version	11
8 PLDM for File Transfer Concepts	12
8.1 File Metadata	12
8.2 File Transfer	13
8.3 File Discovery, Hierarchy and Identity Semantics	13
8.3.1 Semantics	13
8.3.2 File Types and Classification	14
8.3.3 File and Directory Discovery	15
8.3.4 File System Hierarchy Discovery	15
8.4 RequestMaxPoll Usage	15
8.5 RequestCI Usage	16
8.6 DfOpenExclusive Usage	16
8.7 File Zero Length	16
8.7.1 ClientZeroLengthOnly Usage	16
8.8 DSP0248 PLDM for Platform Monitoring and Control Specification Relationship	17
8.8.1 The File Descriptor Data Model	18
8.8.2 Required File Sensors	18
8.8.3 File Size Monitoring Sensor	19
8.8.4 File Size Monitoring Sensor Thresholds	20
8.8.5 Device File State Sensor	20
8.8.6 Sensor and File Transfer command interaction	21
8.8.7 The Directory Descriptor Data Model	21
8.8.8 File Data Model	23
9 PLDM for File Transfer Commands	26
9.1 DfProperties Command	27
9.2 DfOpen Command	28
9.2.1 DfOpen File Host Pushed requirements	28
9.2.2 DfOpen DfOpenAttribute requirements	28
9.2.3 DfOpen SerialTxFIFO requirements	29
9.2.4 DfOpen FileDescriptors count requirements	29
9.2.5 File Client file exclusivity usage	29
9.3 DfClose Command	31
9.4 DfGetFileAttribute Command	33

9.5 DfSetFileAttribute Command . . . . . 35

9.6 DfHeartbeat Command . . . . . 38

    9.6.1 Implicit File Close . . . . . 38

9.7 Error Completion Codes . . . . . 40

9.8 DfRead (DSP0240 MultipartReceive) . . . . . 41

    9.8.1 Serial FIFO type file characteristics . . . . . 41

    9.8.2 DfRead command details . . . . . 42

9.9 DfFIFOSend (DSP0240 MultipartSend) . . . . . 45

10 ANNEX A (informative) Change Log . . . . . 47

11 ANNEX B (informative) Sensor Threshold Event Examples . . . . . 48

12 ANNEX C (informative) File PDR FileClassification FileCapabilities Examples . . . . . 50

13 ANNEX D (informative) PLDM for File Transfer Examples . . . . . 53

    13.1 PLDM for File Transfer initialization example . . . . . 53

    13.2 Regular log file read . . . . . 57

    13.3 Polled Serial Log Read . . . . . 63

    13.4 Pushed Serial Log Read . . . . . 66

# 14 **1 Foreword**

---

15 The *Platform Level Data Model (PLDM) for File Transfer Specification* (DSP0242) was prepared by the Platform Management Communications Infrastructure (PMCI) Work Group.

16 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>.

## 17 **1.1 Acknowledgments**

---

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

- Jeff Wolford – Hewlett Packard Enterprise (Co-Editor)
- Patrick Schoeller – Intel Corporation (Co-Editor)
- Patrick Caporale – Lenovo
- Jim Harford – Broadcom, Inc.
- Yuval Itkin – NVIDIA Corporation
- Tom Joseph – IBM
- Justin King – IBM
- Deepak Kodihalli – NVIDIA Corporation
- Eliel Louzoun – Intel Corporation
- David Rudy – Oracle
- Bill Scherer – Hewlett Packard Enterprise
- Hemal Shah – Broadcom, Inc.
- Supreeth Venkatesh – Advanced Micro Devices

## 19 **2 Introduction**

---

20 The *Platform Level Data Model (PLDM) for File Transfer Specification* defines messages and data structures used for transferring files between PLDM termini, within a PLDM subsystem. Mechanisms to discover files and their metadata are also defined.

### 21 **2.1 Document conventions**

---

22 Refer to [DSP0240](#) for conventions, notations, and data types that are used across the PLDM specifications.

## 23 3 Scope

---

24 This specification describes messages and data structures used to transfer files between PLDM termini, within a PLDM subsystem. It describes mechanisms for the following purposes:

- Discovery of files and directories available on a PLDM terminus for transfer via the File Transfer specific PLDM PDR Repository entries
- Discovery of the file and directory metadata via PLDM PDR entries and File Transfer specific sensors
- Reading *Regular* and *SerialTxFIFO* type files

25 This specification describes the expectations and requirements on PLDM termini that take part in file transfer. The use cases around file transfer, content, and format of the files, are out of scope for this specification. This specification does not specify whether a given system is required to implement that capability. However, if a system does support file transfers over PLDM or other functions described in this specification, the specification defines the requirements to access and use those functions over PLDM. Portions of this specification rely on information and definitions from other specifications, that are identified in the [Normative references](#) clause. Four 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.
- DMTF [DSP0248](#) — PLDM for Platform and Monitoring & Control provides details on file and state sensors, and the file and directory PLDM PDR structures
- DMTF [DSP0249](#) — PLDM State Set Specification provides the definition of the FILE State Sensor

26 The goal of this specification is to model the discovery and access semantics on the industry standard [ISO C Language FILE Library](#) and enable easier and faster adoption. The [ISO C Language FILE Library](#) semantics, such as open, read, and close, are expected to be familiar to the reader. Additionally, to the extent possible, [DSP0240](#) multipart transfers and existing PLDM capabilities including PLDM sensor-based event notifications are leveraged.

27 Both flat (no directories) and hierarchical directory-based file organization are supported.

28 The following are out of scope of this specification:

- Creation of files or directories by a device besides the [File Host](#)
- Direct writes to the File Host

## 29 4 Normative references

---

30 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.

31 DMTF DSP0240 *Platform Level Data Model (PLDM) Base Specification* 1.2 [https://www.dmtf.org/standards/  
published\\_documents/DSP0240\\_1.2.pdf](https://www.dmtf.org/standards/published_documents/DSP0240_1.2.pdf)

32 DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes* 1.4 [https://www.dmtf.org/standards/  
published\\_documents/DSP0245\\_1.4.pdf](https://www.dmtf.org/standards/published_documents/DSP0245_1.4.pdf)

33 DMTF DSP0248, *Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification* 1.3  
[https://dmtf.org/sites/default/files/standards/documents/DSP0248\\_1.3.pdf](https://dmtf.org/sites/default/files/standards/documents/DSP0248_1.3.pdf)

34 DMTF DSP0249, *Platform Level Data Model (PLDM) State Set Specification* 1.2 [https://dmtf.org/sites/default/files/  
standards/documents/DSP0249\\_1.2.pdf](https://dmtf.org/sites/default/files/standards/documents/DSP0249_1.2.pdf)

35 DMTF DSP1001, *Management Profile Specification Usage Guide* 1.1 [https://www.dmtf.org/standards/  
published\\_documents/DSP1001\\_1.1.pdf](https://www.dmtf.org/standards/published_documents/DSP1001_1.1.pdf)

36 DMTF DSP4014, *DMTF Process for Working Bodies* 2.13 [https://www.dmtf.org/sites/default/files/standards/  
documents/DSP4014\\_2.13.pdf](https://www.dmtf.org/sites/default/files/standards/documents/DSP4014_2.13.pdf)

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

38 IETF RFC3629, *UTF-8, a transformation format of ISO 10646* November 2003 <https://www.ietf.org/rfc/rfc3629.txt>

39 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>

40 ISO/IEC 9899:2018, *Information technology - Programming languages - C* <https://www.iso.org/standard/74528.html>



## 41 5 Terms and definitions

---

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

43 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.

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

45 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.

46 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.

47 **Device File State Sensor** PLDM State Set Sensor *Device File (68)* PDR [DSP0249 PLDM State Set](#) used to report possible file states. See [Device File State Sensor](#) for requirements.

48 **File Client** A PLDM Terminus that can receive files from a File Host

49 **File Host** A PLDM Terminus that has a PLDM File Repository and enables a File Client to receive files from the File Host.

50 **File PDR** File Descriptor Platform Descriptor Record (PDR) as defined in [DSP0248 PLDM for Platform Monitoring and Control Specification](#)

51 **File Size** Number of bytes returned by the File Size Monitoring Sensor representing the current length of the associated file or the *File PDR FileMaximumSize* if the associated File Size Monitoring Sensor does not exist.

52 **File Size Monitoring Sensor** A Compact or Numeric sensor PDR (see [DSP0248 PLDM for Platform Monitoring and Control](#)) used to report the current file size in bytes returned by the PLDM GetSensorReading command (see [DSP0248](#)). See [File Size Monitoring Sensor](#) for requirements.

53 **NegotiatedInterval** The maximum negotiated time interval in milliseconds to be used between commands issued by the File Client. See [DfHeartbeat](#) for requirements.

## 54 **6 Symbols and abbreviated terms**

---

55 Refer to [DSP0240](#) for symbols and abbreviated terms that are used across the PLDM specifications. For the purposes of this document, the following additional symbols and abbreviated terms apply.

56 **EAR** Entity Association PDR as defined in [DSP0248 PLDM for Platform Monitoring and Control](#)

57 **FIFO** First in, first out

58 **IANA** Internet Assigned Numbers Authority

59 **OEM** Original Equipment Manufacturer

60 **PDR** Platform Descriptor Record as defined in [DSP0248 PLDM for Platform Monitoring and Control Specification](#)

## 61 **7 PLDM for File Transfer version**

---

62 The version of this Platform Level Data Model (PLDM) for File Transfer Specification shall be 1.0.0 (major version number 1, minor version number 0, update version number 0, and no alpha version).

## 63 8 PLDM for File Transfer Concepts

---

64 This section describes the key concepts of the File Transfer model and outlines expectations on PLDM termini that implement this specification. This section also describes the multipart transfer partnership with the [DSP0240 PLDM Base Specification](#) and [DSP0248 PLDM for Platform Monitoring and Control](#) specifications.

65 The PLDM for File Transfer specification is modeled after the [ISO C Language](#) FILE Library commands but adding the prefix of “Df” (Device File) to the Open, Read, and Close commands. The DfRead command adds an optional offset to implement a Seek and Read styled command.

66 This PLDM specification is part of the PLDM protocol suite and depends on the [DSP0240](#) Discovery and Multiple Part (Multipart) transfer commands, the [DSP0248](#) Platform Descriptor Records (PDR) that includes File Descriptor, Numeric Sensor, State Sensor, and Entity Association (EAR) PDRs. There are also [DSP0248](#) commands to interact with the Platform Descriptor Records.

67 The PLDM Initialization Agent discovers the PLDM for File Transfer support including supported specification version and commands as defined in [DSP0240](#). The data model definition for a file and an optional associated directory is represented by the *File PDR* with hierarchy expressed with EARs. The data model provides static (meta) data in the *File PDR* and dynamic data using numeric and state sensors.

---

68  **NOTE:** The following list presents an example of a typical PLDM for File Transfer data flow:

- The File Client issues the NegotiateTransferParameters from [DSP0240](#) with the File Host.
- The File Client retrieves the list of files, dynamic attributes (sensors), and optional directories from the File Host [DSP0248](#) specification defined PDR Repository.
- The File Host may generate events using the [DSP0248](#) PlatformEventMessage command. The File Client may choose to receive events using the [DSP0248](#) specification SetEventReceiver command.
- The File Client issues the DfOpen command, using the *FileIdentifier* from a *File PDR*, to the File Host who returns a session *FileDescriptor* used in applicable PLDM for File Transfer commands.
- The data transfer command from the File host is performed using a DfRead command, a logical construction mapped to the [DSP0240](#) MultipartReceive command.
- Upon completing the DfRead command, the File Client either issues a DfClose command or issues a DfHeartbeat command.

---

### 69 8.1 File Metadata

70 The static file metadata can be obtained by retrieving the appropriate *File PDR*. Dynamic file metadata, such as [File Size](#), can be obtained by reading the [File Size Monitoring Sensor](#). The methods of retrieving the *File PDR* and reading the [File Size Monitoring Sensor](#) are defined in [DSP0248](#).

---

## 71 8.2 File Transfer

---

72 A PLDM requester, typically a platform Baseboard Management Controller, is the originator of PLDM for File Transfer initiated by the [DfOpen](#) command and performs the role of the File Client. A PLDM Terminus that responds to the DfOpen command performs the role of the File Host. The characteristics of these roles are:

- 73 • **File Host** — A PLDM Terminus that:
- Creates, modifies, deletes files
  - Presents a listing of files to a File Client using the [DSP0248](#) PDR Repository
  - Transfers files to a File Client using the mechanisms defined in this specification
- 74 • **File Client** — A PLDM Terminus that:
- Initiates a file transfer session to a File Host
  - Receives files from a File Host using the mechanisms described in this specification
  - Controls specific behavior such as preservation

## 75 8.3 File Discovery, Hierarchy and Identity Semantics

---

76 This section describes the terminology and semantics used by this specification as they relate to the [ISO C Language](#) FILE library semantics.

### 77 8.3.1 Semantics

- 78 • **File**
- A file is an entity identified by a *File PDR* and has the *EntityType* set to *Device File*
  - A file is a physical object that consumes storage space. The allocated storage may be volatile or non-volatile
  - The *File PDR* has a field, *FileIdentifier*, that is a single unique numeric value representing the file name within the File Host hierarchy. The file name is a defined field in the *File PDR*.
  - A file may be associated to a directory by the tuple: *ContainerID*, *EntityType*, *EntityInstanceNumber*. If the File Host establishes a directory hierarchy, the directory association to its file (members) is constructed using an EAR.
- 79 • **Directory**
- A directory is a logical object that associates files within its hierarchy
  - A directory is an entity identified by a *File PDR* and has the *EntityType* set to *Device File Directory*
  - The *File PDR* has a field, *FileIdentifier*, that is a single unique numeric value representing the directory name within the File Host hierarchy. The directory name is a defined field in the *File PDR*.

- The directory shall be a PLDM Container of a PLDM EAR that associates files into its hierarchy

80 • **FileIdentifier**

- A unique numeric value, obtained from the *File PDR*, that represents the file name or the directory name within the File Host hierarchy
- The *FileIdentifier* and *FileName* fields are part of the same *File PDR*.
- The *FileIdentifier* is used (instead of the *FileName* (string) field) for specific FILE type commands such as *DfOpen*, *DfGetFileAttribute*, and *DfSetFileAttribute*.
- When a Device File Directory *FileIdentifier* is a parameter to a *DfOpen*, *DfGetFileAttribute*, or *DfSetFileAttribute* command, an *INVALID\_FILE\_IDENTIFIER CompletionCode* shall be returned.

81 • **FileDescriptor**

- The *FileDescriptor* is returned from the *DfOpen* command and represents a session to a specific file.
- Similar to the [ISO C Language](#) FILE Library functions, the *FileDescriptor* is the session identifier for *DfRead*, *DfHeartbeat*, and *DfClose* commands defined in this specification, similar to the FILE object returned from *fopen()*.
- The *FileDescriptor* is the *DfRead* command (PLDM MultipartReceive command) *TransferContext* value to identify the file and the session owning the data transfer.

82 **8.3.2 File Types and Classification**

83 Files are physical entities that have capabilities and classifications. There are also dynamic attributes that may be set by the File Client executing the *DfSetFileAttribute* command if supported by the File Host. Examples of static file capabilities that are normatively defined in the *File PDR* are: Exclusive Open, File Truncation / Wrapping, and File Classification.

84 This specification, in collaboration with the [DSP0248 PLDM for Platform Monitoring and Control](#) specification, is recommending an industry-conventional file (data) classification to allow the File Client to understand the type of contained file data.

85  **NOTE:** [Table 1](#) has some examples of file (data) classifications.

86 **Table 1 — FileClassification Examples**

<i>FileClassification</i>	<i>Definition</i>
<i>BootLog</i>	Typically holds device initialization data (events) but has no additional entries after initialization completes
<i>SerialTxFIFO</i>	Typically removes the data after successful transfer to the receiver or if the FIFO overflows
<i>DiagnosticLog</i>	Typically a variable length file where data can be appended until maximum storage limit is reached.

<i>FileClassification</i>	<b>Definition</b>
<i>CrashDumpFile</i>	A fixed-length file (instance) after creation, written one time with no growth per crash event, typically containing diagnostic data
<i>FRUDataFile</i>	A fixed-length file that stores Field Replaceable Unit (FRU) data typically found on add-in adapters
<i>OtherLog</i>	A file classification that implies growth (appends) for new event (data).
<i>OtherFile</i>	A file classification that implies a “write data once” with no growth after event (data) written.

87

### 88 **8.3.3 File and Directory Discovery**

89 Files and Directories are discovered by collecting the *File PDRs* with *EntityType* set to *Device File* or *Device File Directory*. The *File PDR* holds static (meta) data including the hierarchy, identity, and static maximum file size. When a File Host creates or deletes a file, the *GetPDRRepositoryInfo* update time is modified, the *GetPDRRepositorySignature* is different, and a *PldmPDRRepositoryChgEvent* may be signaled if PLDM Events are enabled.

90 The expectation is that file creation and deletion activity is not frequent. The recommended use case is for the File Host to create expected files (with PDR) but not write the data until required. The File Client may periodically poll the [File Size Monitoring Sensor](#) for the current file size, or the File Client may enable PLDM Events for the [Device File State Sensor](#) to be alerted when a file has changed. The file does not have to be open for this activity because this is normative [DSP0248](#) supporting functionality that is foundational to this specification.

### 91 **8.3.4 File System Hierarchy Discovery**

92 The file system hierarchy of a File Host is learned through the PDRs and EARs. If the File Host implements a hierarchy of directories to contain files, then the File Host shall implement the directory structure using the EAR data model. The *ContainerEntityContainerID* shall be the directory identifier and all PDRs whose *ContainerID* matches the directory identifier value shall be contained within the specified directory.

93 This specification’s recommended implementation is to create the PDRs for known file types, which allows the File Client to collect the hierarchical data during PLDM Device Initialization.

## 94 **8.4 RequestMaxPoll Usage**

95 The *DfGetFileAttribute* command *RequestMaxPoll* attribute is intended to be used when a File Client is interacting with a File Host implementing a limited buffer *SerialTxFIFO*. The *RequestMaxPoll* represents the maximum time the File Client may take between reading the PLDM *File Size Monitoring Sensor* and the *DfRead* command or between two *DfRead* commands of this file before data is lost either to truncating or wrapping.

## 96 8.5 RequestCI Usage

---

97 The DfGetFileAttribute command *RequestCI* attribute is intended to be used when a File Client is interacting with a file that does not change in size when updated and the File Client needs to know if the file has changed since the last time it was read.

98 To do this, the File Client would retain the previous *ChangeIndicator* value and compare it to the new *ChangeIndicator* value where a difference would indicate the file has changed.

## 99 8.6 DfOpenExclusive Usage

---

100 The *DfOpenAttribute DfOpenExclusive*, when combined with *DfOpenReadWrite* set to zero (0) of the DfOpen command, is used to enable the *DfClose* command *ZeroLength* option set to one (1) as described in *DfClose*. The need for the exclusivity is so that the file is not modified by the File Host or by another File Client while current File Client has an active *file session*. When used in conjunction with the *ClientZeroLengthOnly* attribute of the DfSetFileAttribute command, it allows the File Client to control when a file is updated. See the NOTE in *File Zero Length*.

## 101 8.7 File Zero Length

---

102 File zero length overview:

- The choice of allowing a *File Size* to be set to zero (0) by the File Client issuing a DfClose command *ZeroLength* option set to one (1) is optional on a file by file basis by the File Host and indicated by the *File PDR FileCapabilities FcZeroLengthPermitted* bit set to one (1).
- If the File Host allows the File Client to issue a DfClose Command *ZeroLength* option set to one (1) on a file, the File Client may request that only the File Client may modify the file by setting the *ClientZeroLengthOnly* attribute of the DfSetFileAttribute command to one (1) when there is no active file session established.

### 103 8.7.1 ClientZeroLengthOnly Usage

---

104  NOTE

105 One use of the *ClientZeroLengthOnly* is to make sure a *CrashDumpFile* is not overwritten or deleted by the File Host before the File Client has read the file. Ideally the *CrashDumpFile* is stored in non-volatile memory and is preserved over resets and power cycles as described in *Table 14*.

106 In order to eliminate the race condition of a File Host *CrashDumpFile* generation, File Client reading, File Host deleting and/or overwriting the file, the following example sequence is envisioned:

---



- 107 1. The File Host updates the [File Size Monitoring Sensor](#) representing the current file size to be zero length and the *FcZeroLengthPermitted File PDF FileCapabilities* set to one (1).
- 108 2. Immediately after discovering that the [File Size Monitoring Sensor](#) representing the *CrashDumpFile* PDR is zero (0) length, the File Client sets the *ClientZeroLengthOnly* attribute of the *DfSetFileAttribute* command to one (1) without opening the file.
- 109 3. With the *ClientZeroLengthOnly* attribute set, the File Host is allowed to update the *CrashDumpFile* one time to cause the size to go from zero (0) to the *CrashDumpFile* final size.
- 110 4. When the File Host generates a *CrashDumpFile*, it now:
  - i. Updates *File Size* with the current file size
  - ii. Updates the [Device File State Sensor](#) to *File is Updated*
  - iii. Makes no updates to the *CrashDumpFile File PDR*
- 111 5. The File Client can be notified that a *CrashDumpFile* has been populated in several different ways:
  - i. Register for events from the [File Size Monitoring Sensor](#)
  - ii. Register for events from the [Device File State Sensor](#)
  - iii. Poll the [File Size Monitoring Sensor](#) looking for a nonzero *File Size* value
  - iv. Poll the [Device File State Sensor](#) looking for *File is Updated*
- 112 6. The File Client proceeds with a *DfOpen* with exclusivity and *DfRead* command sequence to retrieve the *CrashDumpFile*.
- 113 7. Since the File Client has opened the file exclusively, it can now immediately issue a *DfClose* command *ZeroLength* option set to one (1) to minimize the possibility of locking out the File Host from generating a new *CrashDumpFile* if so needed.
- 114 8. At this point the File Client can go back to waiting on a *CrashDumpFile* notification and the File Host may generate another *CrashDumpFile* if so needed.

---

## 115 8.8 DSP0248 PLDM for Platform Monitoring and Control Specification Relationship

116 This section describes the Platform Level Data Model (PLDM) used within the context of this specification. The specification declares normative usage of PLDM objects such as Platform Descriptor Records (PDRs) and specific value assignments within the data model. The reader should refer to other PLDM specifications for objects and fields not explicitly stated in this specification.

117 **8.8.1 The File Descriptor Data Model**

118 *File PDR* requirements:

- Every File shall have a *File PDR* that provides static metadata such as the (file) object maximum size.
- Every File shall have an *EntityType* set to *Device File*.

119 **8.8.2 Required File Sensors**

120 [Table 2](#) describes the different file characteristics.

121 **Table 2 — File Characteristic Definitions**

File Characteristic	Definition
<i>Static File</i>	READ ONLY after creation; the contents do not change; no updates or appends; does not support the DfClose command <i>ZeroLength</i> option; file size is equal to the value of the <i>File PDR FileMaximumSize</i> field
<i>Fixed Length File</i>	READ and MODIFY after creation but no append; the size does not change but may be updated; does not support the DfClose command <i>ZeroLength</i> option; file size is equal to the value of the <i>File PDR FileMaximumSize</i> field
<i>Regular File</i>	READ, MODIFY, or APPEND after creation, including the optional DfClose <i>ZeroLength</i> option
<i>SerialTxFIFO</i>	A FIFO file whose file size may grow or shrink

122  **NOTE:** [Table 3](#) describes the sensor usage for different file characteristics.

123 **Table 3 — Sensor Usage for Different File Characteristics**

File Characteristic	Usage
<i>Static File</i>	The file is always at maximum size, so having a <i>Device File State Sensor File is at Maximum Size State</i> is meaningless.  File is a static size, so any <i>File Size Monitoring Sensor</i> thresholds are meaningless.
<i>Fixed Length File</i>	The <i>Device File State Sensor</i> is used to indicate and/or notify of a file update.
<i>Regular File</i>	The <i>FatalHigh</i> threshold is used to allow overflow detection.  <i>Device File State Sensor</i> is used to indicate and/or notify of a file update.

File Characteristic	Usage
<i>SerialTxFIFO</i>	<p>The <i>File Size Monitoring Sensor WarningHigh</i> threshold allows the File Client to be notified it is not retrieving / accepting data faster than the data being written to the FIFO.</p> <p>The <i>File Size Monitoring Sensor FatalHigh</i> threshold allows the File Client to be notified that data has overflowed.</p> <p>The <i>Device File State Sensor</i> is used to indicate and/or notify of an overflow condition.</p>

124 [Table 4](#) describes the sensors and thresholds required for each file characteristic.

125 **Table 4 — File Sensors and Thresholds Support**

File Characteristic	File Size Monitoring Sensor Support	File Size thresholds Support	Device File State Sensor Support	Device File State Support
<i>Static File</i>	Should	Should not	Should	Should not support <i>File is at Maximum File State</i>
<i>Fixed Length File</i>	Should	Should not	Shall	Should not support <i>File is at Maximum File State</i>
<i>Regular File</i>	Shall	Should	Shall	Should support all defined states
<i>SerialTxFIFO</i>	Shall	Shall	Shall	Should not support <i>File is Updated</i> or <i>File is not updated</i>

126 **8.8.3 File Size Monitoring Sensor**

127 File Host *File Size Monitoring Sensor* requirements:

- The *File Size Monitoring Sensor* shall be implemented as a Compact or Numeric sensor PDR used to report the current file size in bytes in the PresentReading field and monitor file size changes. Optionally, the *File Size Monitoring Sensor* may be used to generate PLDM events (using threshold limits), either by the File Host as a default or by an explicit [DSP0248](#) SetNumericSensorEnable command. The File Client should also send a [DSP0248](#) SetEventReceiver command to enable reception of the event messages.
- The *File Size Monitoring Sensor* shall match the monitored *File PDR EntityType*, *EntityInstance*, and *ContainerID* fields to establish its association to the monitored file.
- The *File Size Monitoring Sensor BaseUnit* shall be set to *Bytes*.
- The *File Size Monitoring Sensor UnitModifier* shall be set to zero (0).
- If the *File Size Monitoring Sensor* is a Numeric Sensor type, then:
  - *RateUnit* shall be set to *None*.
  - *Offset* shall be set to zero (0).

- *Resolution* shall be set to 1.00 (real32 data type; for this, see [DSP0240](#)).
- If the *File Size Monitoring Sensor* is a Compact Numeric Sensor type, then *OccurrenceRate* shall be set to *No Occurrence Rate*.

128 File Client *File Size* requirements:

- If the *File PDR* does not have an associated *File Size Monitoring Sensor*, then the *File Size* is the number of bytes indicated by the *File PDR FileMaximumSize* field; otherwise the *File Size* is the value returned by the *GetSensorReading* command of the associated *File Size Monitoring Sensor*.

## 129 8.8.4 File Size Monitoring Sensor Thresholds

- 130 • If a *File Size Monitoring Sensor* supports thresholds, then:
- The File Host should set *WarningHigh* and/or *CriticalHigh* thresholds based on the file type and the growth rate.
  - The File Host should set the *FatalHigh* threshold equal to the *File PDR FileMaximumSize* field.
  - The File Host should support the [DSP0248](#) *SetSensorThresholds* command for the *WarningHigh* and *CriticalHigh* thresholds to allow the File Client to adjust priority and buffering; this is critical for *SerialTxFIFO FileClassification* files.
  - The File Client shall not adjust the *FatalHigh* threshold greater than the *File PDR FileMaximumSize*. If the File Client attempts to adjust the *FatalHigh* threshold greater the *File PDR FileMaximumSize*, the File Host shall return an `ERROR_INVALID_DATA CompletionCode` as described in [DSP0240](#).

## 131 8.8.5 Device File State Sensor

132 *Device File State Sensor* requirements:

- 133 • The *Device File State Sensor* shall be implemented as a PLDM State Sensor PDR to report specific file states including file updates without a file size change (such as an inner record modification). The *Device File State Sensor* may be used to generate PLDM events or an explicit [DSP0248](#) *SetStateSensorEnables* command.
- 134 • The *Device File State Sensor* shall match the monitored *File PDR EntityType*, *EntityInstance*, and *ContainerID* fields to establish its association to the monitored file.
- 135 • The *Device File State Sensor* shall implement the [DSP0249 PLDM State Set](#) Specification State Set *Device File (68)*.
- 136 • For *SerialTxFIFO FileClassification* files:
- If using the *Polled* access method, the File Client should prioritize reading the *File Size Monitoring Sensor* based on the value from the *DfGetFileAttribute* command *RequestMaxPoll*.
- 137 • For all other *FileClassification* files:

- 138 ◦ The File Host should only set the *Device File State Sensor* to *File is Updated* when the following Device File States are not valid or not supported:
- *File is at Maximum State*
  - *File Data has Truncated*
  - *File Data has Wrapped*

139 **8.8.6 Sensor and File Transfer command interaction**

140 [Table 5](#) lists the interactions between the *File Size Monitoring Sensor*, *Device File State Sensor*, and File Transfer (PLDMType seven (7)) commands.

141 **Table 5 — Sensor File Transfer Command Interactions**

<i>FileClassification</i>	<i>File Transfer Command</i>	<i>Sensor Interaction</i>
<i>SerialTxFIFO</i>	DfOpen DfClose DfGetFileAttribute DfSetFileAttribute DfHeartbeat DfProperties	none
<i>SerialTxFIFO</i>	DfRead	File Host shall update the <i>File Size Monitoring Sensor</i> and the <i>Device File State Sensor</i> .
<i>BootLog</i> <i>DiagnosticLog</i> <i>CrashDumpFile</i> <i>SecurityLog</i> <i>FRUDataFile</i> <i>OtherLog</i> <i>OtherFile</i>	DfClose <i>ZeroLength=1</i>	If implemented, the File Host shall update the <i>File Size Monitoring Sensor</i> and the <i>Device File State Sensor</i> .
<i>BootLog</i> <i>DiagnosticLog</i> <i>CrashDumpFile</i> <i>SecurityLog</i> <i>FRUDataFile</i> <i>OtherLog</i> <i>OtherFile</i>	All commands except DfClose <i>ZeroLength=1</i>	none
<i>OEM</i>	All commands	OEM specific

142 **8.8.7 The Directory Descriptor Data Model**

143 Requirements for Directory *File PDR*:

- The directory shall have a *File PDR*.
- The *File PDR* field *EntityType* shall be set to *Device File Directory*.
- The *File PDR* field *FileCapabilities* shall be set to zero (0).
- The *File PDR* field *FileVersion* shall be set to unversioned (0xFFFFFFFF).
- The *File PDR* field *FileMaximumSize* shall be set to the special value 0xFFFFFFFF.

144 A directory shall be represented as an EAR, such that the directory *File PDR ContainerID* shall be the directory EAR *ContainerID*.

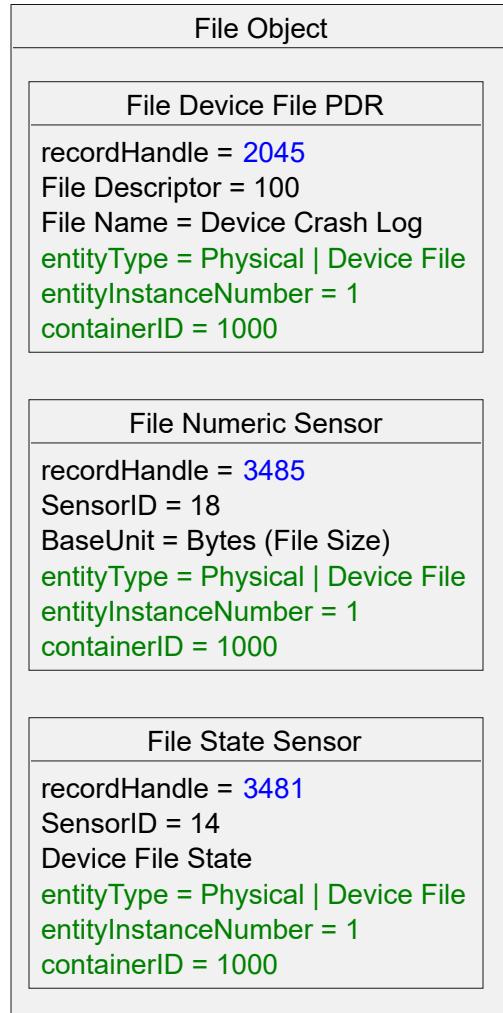
145 Requirements for EAR representation of the Directory:

- The EAR *AssociationType* shall always be set to *LogicalContainment*.
- The EAR fields: *ContainerEntityType* and *ContainerEntityInstanceNumber* shall match the associated directory *File PDR EntityType* and *EntityInstance* values.
- The EAR field *ContainerEntityContainerID* is recommended to be set to the special value *System* or to the *ContainerID* of a superior directory.
- All files subordinate to a directory shall have their *File PDR EntityType*, *EntityInstance*, and *ContainerID* fields listed in the directory's EAR *Contained Entity Identification Information* section.

146 See [Figure 1](#) for an example of the implicit association of a file object with its associated sensors, using the PDR association fields *EntityType*, *EntityInstance*, and *ContainerID*.

147 **8.8.8 File Data Model**

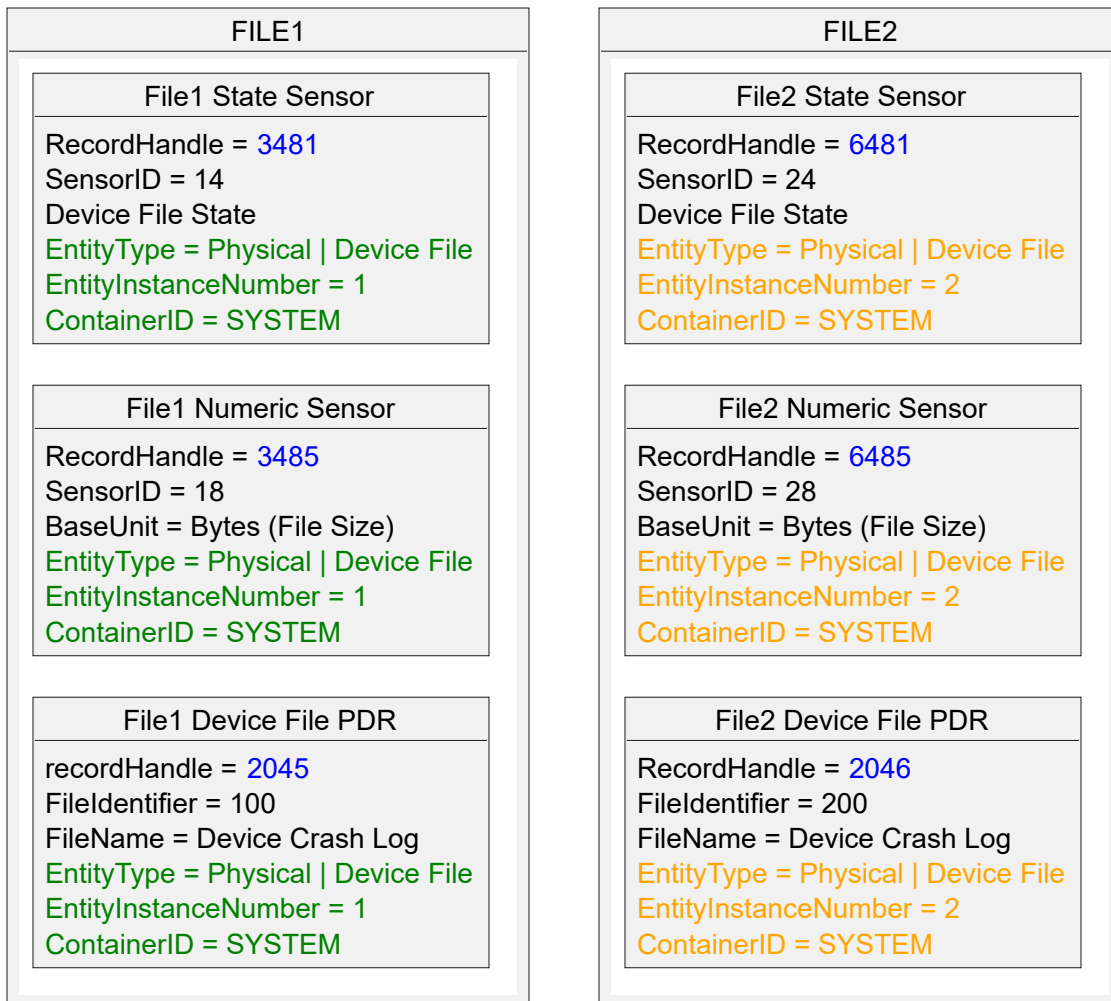
148 In [Figure 1](#) the numeric and state sensors associated with the file match the *EntityType*, *EntityInstance*, and  
 149 *ContainerID* fields of the *File PDR*.



150 **Figure 1 — PLDM for File Transfer File Data Model Implicit Association Example**

151 Figure 2 shows a flat file sensor usage example.

152

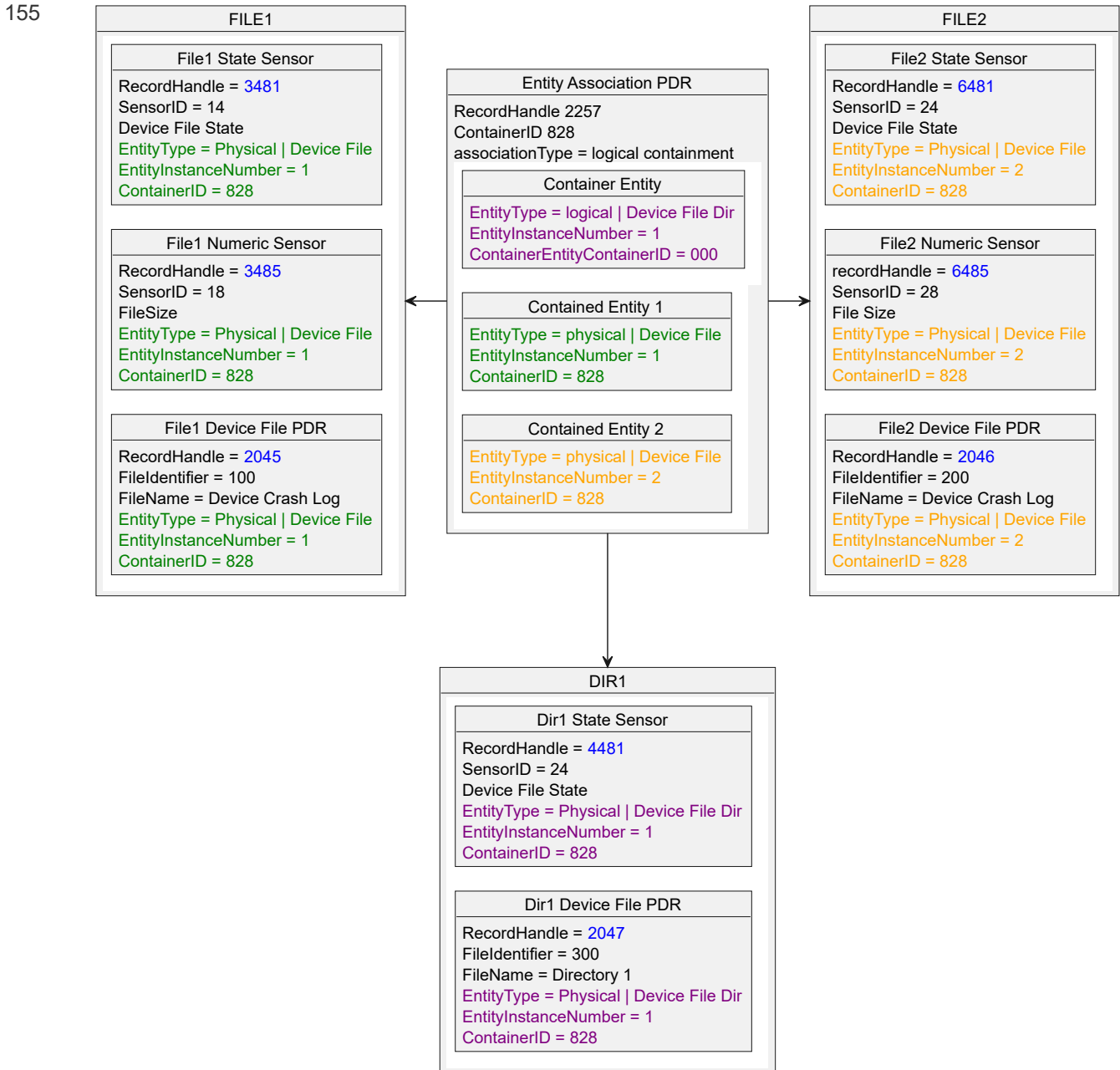


153

Figure 2 — PLDM for File Transfer Flat File Sensor Usage Example



154 Figure 3 shows a Directory EAR sensor usage example



156 **Figure 3 — PLDM for File Transfer Directory EAR Sensor Usage Example**

## 157 9 PLDM for File Transfer Commands

158 This section describes the commands that shall be used for File Transfer. [Table 6](#) consists of the codes assigned to commands. These commands have their own PLDM message type that is defined in [DSP0245](#).

159 **Table 6 — PLDM for File Transfer Command Codes**

Command	Code Value	File Host support <sup>3</sup>	File Client support <sup>4</sup>
<a href="#">DfOpen</a>	0x01	Required	Required
<a href="#">DfClose</a>	0x02	Required	Optional
<a href="#">DfHeartbeat</a>	0x03	Optional	Conditional
Reserved	0x04-0x0F		
<a href="#">DfProperties</a>	0x10	Required	Optional
<a href="#">DfGetFileAttribute</a>	0x11	Optional	Optional
<a href="#">DfSetFileAttribute</a>	0x12	Optional	Optional
Reserved	0x13-0x1F		
<a href="#">DfRead</a> <a href="#">MultipartReceive</a>	0x20 <sup>1</sup>	Required	Required
<a href="#">DfFIFOSend</a> <a href="#">MultipartSend</a>	0x21 <sup>1</sup>	Conditional <sup>2</sup>	Optional
Reserved	0x22-0xFF		

160 <sup>1</sup> This command value is only to support the [DSP0240](#) GetPLDMCommands command for discovery and is mapped to [DSP0240](#) PLDM Multipart Transfer.

<sup>2</sup> If *File PDR* has both *FileCapabilities DataType* set to one (1) (*Streaming FIFO*) and *Pushed* set to one (1), then the File Host shall support the DfFIFOSend command.

<sup>3</sup> The File Host is the responder except for the DfFIFOSend command where it is the requester.

<sup>4</sup> The File Client is the requester except for the DfFIFOSend command where it is the responder.

161 For Optional or Conditional command requirements, see the individual command descriptions.

162 **9.1 DfProperties Command**

163 The File Client issues a DfProperties command, as described in [Table 7](#) to list the File Transfer specific capabilities of the File Host.

164 If more than one bit is set in *DfPropertyAttribute*, or a specified *DfPropertyAttribute* is not supported, then the File Host shall return an `INVALID_DF_ATTRIBUTE CompletionCode`.

165 **Table 7 — DfProperties Command Format**

Byte	Type	Request Data
0:3	bitfield32	<b>DfPropertyAttribute</b> 1b = Request the specified <i>DfPropertyAttribute</i> 0b = <i>DfPropertyAttribute</i> not requested [0] — <i>MaxConcurrentMedium</i> [1] — <i>MaxFileDescriptors</i> [2:31] — Reserved
Byte	Type	Response Data
0	enum8	<b>CompletionCode</b> Possible values: { <code>PLDM_BASE_CODES</code> , <code>INVALID_DF_ATTRIBUTE</code> } See <a href="#">Table 16</a> for values.
1:4	uint32	<b>DfPropertyAttributeValue</b> See <a href="#">Table 8</a> .

166 **Table 8 — *DfPropertyAttributeValue* Definition**

<i>DfPropertyAttributeValue</i>	Definition
<i>MaxConcurrentMedium</i>	The maximum number of mediums the File Host support. The returned <i>MaxConcurrentMedium</i> shall be one (1) for this specification.
<i>MaxFileDescriptors</i>	The total number of File Descriptors the File Host supports.

## 167 9.2 DfOpen Command

---

- 168 The File Client issues a DfOpen command to establish a file session between the File Client and a specific file. The DfOpen command, as described in [Table 9](#), uses a *File PDR FileIdentifier* field to access the specific file.
- 169 The *FileDescriptor* shall be unique to the File Host and may be used by the File Host to track how many File Clients have a file open.
- 170 The returned *FileDescriptor* is used by the File Client for subsequent DfRead, DfHeartbeat, and DfClose commands.
- 171 The DfOpen command only supports the *File PDR* with the entity type set to *Device File*. If the File Host receives a DfOpen command with a *FileIdentifier* associated to a *File PDR* with the entity type set to *Device File Directory*, then the File Host shall return a `DFOPEN_DIR_NOT_ALLOWED` *CompletionCode*.
- 172 If the file specified by the *File PDR FileIdentifier* does not exist, the File Host shall return an `INVALID_FILE_IDENTIFIER` *CompletionCode*.
- 173 If the file specified by the *File PDR FileIdentifier* exists, but the File Host temporarily cannot return a *FileDescriptor*, then the File Host shall return an `UNABLE_TO_OPEN_FILE` *CompletionCode*.

### 174 9.2.1 DfOpen File Host Pushed requirements

- 175 *DfOpenPolledPushed* set to one (1) (*Pushed*) is only supported if *DfOpenRegFIFO* is set to one (1) (*FIFO*).
- 176 If the File Client sets *DfOpenPolledPushed* attribute to one (1) (*Pushed*) in the DfOpen command and the DfOpen command is successfully completed, the File Client shall be able to immediately receive the start of a DfFIFOSend command and no DfRead command is required or allowed.

### 177 9.2.2 DfOpen DfOpenAttribute requirements

- 178 File Client / File Host *DfOpenAttribute* requirements:
- 179 • If the File Client issues a DfOpen command with an invalid combination of *DfOpenAttribute* or unsupported *DfOpenAttribute* for the requested *FileIdentifier*, then the File Host shall return the `INVALID_DF_ATTRIBUTE` *CompletionCode*.
  - 180 • If the File Host is temporarily unable to establish exclusive ownership of the requested *FileIdentifier* with the *DfOpenAttribute DfOpenReadWrite* set to zero (0) and *DfOpenAttribute DfOpenExclusive* set to one (1) (*Exclusive*), and if exclusive ownership is allowed (that is, if *File PDR FileCapabilities ExReadOpen* is set to one (1)), then the File Host shall return the `EXCLUSIVE_OWNERSHIP_NOT_AVAILABLE` *CompletionCode*.
  - 181 • If the File Client successfully completes a DfOpen command with *DfOpenAttribute DfOpenReadWrite* set to zero (0) and *DfOpenAttribute DfOpenExclusive* set to one (1) (*Exclusive*), the File Host shall not make any updates, including changing the length of the file represented by the requested *FileDescriptor*.
    - If the File Host cannot support this requirement for this file, then it shall set the *File PDR FileCapabilities*

*ExReadOpen* to zero (0) for this file.

- 182 • See [DfOpen SerialTxFIFO requirements](#) for additional *DfOpenAttribute* requirements when *File PDR FileClassification* equals *DfOpen SerialTxFIFO*.

### 183 9.2.3 DfOpen SerialTxFIFO requirements

184 Requirements when *File PDR FileClassification* equals *DfOpen SerialTxFIFO*:

185 By definition of FIFO (first in, first out), the *DfOpen* of a *SerialTxFIFO* file does not support multiple simultaneous [file sessions](#):

- Setting the *DfOpen* command *DfOpenAttribute DfOpenExclusive* to one (1) (*Exclusive*) is not supported, and the File Host shall return the *INVALID\_DF\_ATTRIBUTE CompletionCode*.
- The *File PDR FileMaximumFileDescriptorCount* shall be set to one (1).
- The *File PDR FileCapabilities ExReadOpen* shall be set to zero (0).

### 186 9.2.4 DfOpen FileDescriptors count requirements

187 If the number of open *FileDescriptors* for a specified *FileIdentifier* would exceed the *File PDR FileMaximumFileDescriptorCount* (see [DSP0248](#)), then the File Host shall return the *MAX\_NUM\_FDS\_EXCEEDED CompletionCode*.

188 If the number of open *FileDescriptors* for the File Host would exceed the *DfProperties* command *MaxFileDescriptors*, then the File Host shall return the *MAX\_NUM\_FDS\_EXCEEDED CompletionCode*.

### 189 9.2.5 File Client file exclusivity usage

190 The File Client should only open a file exclusively for the following reasons:

- The File Client would like to issue a *DfClose* command *ZeroLength* option set to one (1) on this file.
- The *ClientZeroLengthOnly FileCapabilities* is set to one (1) for the requested *FileIdentifier*.

191 Due to the restrictions placed on the File Host with the *DfOpen* command *DfOpenAttribute DfOpenReadWrite* set to zero (0) and *DfOpenAttribute DfOpenExclusive* set to one (1) (*Exclusive*), the File Client should minimize the time the file is opened with the *DfOpenAttribute DfOpenExclusive* set to one (1) (*Exclusive*).

192 If the File Host cannot update a file because the File Client has the file opened exclusively and the data is lost, then the File Host should set the [Device File State Sensor](#) to “File Data has Truncated”.

193 **Table 9 — DfOpen Command Format**

Byte	Type	Request Data
0:1	uint16	<b>FileIdentifier</b>

Byte	Type	Request Data
2:3	bitfield16	<b>DfOpenAttribute</b> [0] — DfOpenReadWrite, 1b = <i>Write</i> (not supported), 0b = <i>Read</i> [1] — DfOpenExclusive, 1b = <i>Exclusive</i> , 0b = <i>Non-exclusive</i> [2] — DfOpenRegFIFO, 1b = <i>Streaming FIFO</i> (Serial FIFO), 0b = <i>Regular</i> [3] — DfOpenPolledPushed, 1b = <i>Pushed</i> , 0b = <i>Polled</i> [4:15] Reserved (0)
Byte	Type	Response Data
0	enum8	<b>CompletionCode</b> Possible values: { PLDM_BASE_CODES, UNABLE_TO_OPEN_FILE, INVALID_FILE_IDENTIFIER, INVALID_DF_ATTRIBUTE, EXCLUSIVE_OWNERSHIP_NOT_ALLOWED, EXCLUSIVE_OWNERSHIP_NOT_AVAILABLE, DFOPEN_DIR_NOT_ALLOWED, MAX_NUM_FDS_EXCEEDED } See <a href="#">Table 16</a> for values.
1:2	uint16	<b>FileDescriptor</b>

## 194 9.3 DfClose Command

195 The DfClose command, as described in [Table 10](#), is used by the File Client to tell the File Host the File Client no longer needs access to a file. After the File Client has successfully completed a DfClose, the File Client no longer needs to issue DfHeartbeat commands for that file.

196 If the File Host returns *CompletionCode* equal to SUCCESS for a DfClose command *ZeroLength* option set to zero (0), then the File Host shall:

1. Close the active open file session
2. Invalidate the *FileDescriptor*

197 If the File Host returns *CompletionCode* equal to SUCCESS for a DfClose command *ZeroLength* option set to one (1), then the File Host shall:

1. Set the *File Size* to zero (0)
2. Close the active open file session
3. Invalidate the *FileDescriptor*

198 If the File Client issues a DfClose command with an invalid or no longer valid *FileDescriptor*, then the File Host shall return an INVALID\_FILE\_DESCRIPTOR *CompletionCode*.

199 Requirements for DfClose command *ZeroLength* option set to one (1):

- A File Client shall successfully establish read exclusivity by completing a DfOpen command with *DfOpenReadWrite* set to zero (0) and *DfOpenExclusive* set to one (1) (*Exclusive*).
- If the File Client has not established read exclusivity, then the File Host shall return an EXCLUSIVE\_OWNERSHIP\_NOT\_ESTABLISHED *CompletionCode* to the requested DfClose command.

200 If a File Host supports the DfClose command *ZeroLength* set to one (1), then:

- if the *FileDescriptor* represents a file that has the *FileCapabilities FcZeroLengthPermitted* bit set to zero (0), then the File Host shall return the *CompletionCode* ZEROLENGTH\_NOT\_ALLOWED.
- if the *FileDescriptor* represents a file that has the *FileCapabilities FcZeroLengthPermitted* bit set to one (1) and the File Host cannot change the file's length to zero (0) at the time of the request, then the File Host shall return the PLDM\_BASE\_CODE *CompletionCode* ERROR\_NOT\_READY.

201 If the *CompletionCode* is not equal to SUCCESS, then the open file session remains active and the *FileDescriptor* remains valid and retains the same value.

202 See [Implicit File Close](#) under the DfHeartbeat command for information on implicitly closing a file.

203 **Table 10 — DfClose Command Format**

Byte	Type	Request Data
0:1	uint16	<b>FileDescriptor</b>

Byte	Type	Request Data
2:3	bitfield16	<b>DfCloseOptions</b> 1b = Closing option requested 0b = Closing option not requested [0] — ZeroLength, 1b = Request the File Host set the File Length to zero (0), 0b = No request [1:15] Reserved (0)
Byte	Type	Response Data
0	enum8	<b>CompletionCode</b> Possible values: {PLDM_BASE_CODES, INVALID_FILE_DESCRIPTOR, ZEROLENGTH_NOT_ALLOWED, EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED} See <a href="#">Table 16</a> for values.



## 204 9.4 DfGetFileAttribute Command

205 The DfGetFileAttribute command, as described in [Table 11](#), is used by the File Client to get specific dynamic attributes of a file.

206 If the *FileIdentifier* requested by the File Client is invalid, the File Host shall return an INVALID\_FILE\_IDENTIFIER *CompletionCode*.

207 If the requested *FileIdentifier AttributeReq* is not supported by the File Host, then it shall return an INVALID\_DF\_ATTRIBUTE *CompletionCode*.

208 If more than one bit is set in *AttributeReq*, then the File Host shall return an INVALID\_DF\_ATTRIBUTE *CompletionCode*.

209 **Table 11 — DfGetFileAttribute Command Format**

Byte	Type	Request Data
0:1	uint16	<b>FileIdentifier</b> This is the <i>FileIdentifier</i> returned in the <i>File PDR</i> for this file (directory).
2:5	bitfield32	<b>FileAttributeReq</b> 1b = Request the specified current attribute status 0b = Attribute status not requested [0] — ClientZeroLengthOnly [1:15] Reserved (0) [16] — RequestCI [17] — ReqMaxPoll [18:31] Reserved (0)
Byte	Type	Response Data
0	enum8	<b>CompletionCode</b> Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIBUTE, INVALID_FILE_IDENTIFIER } See <a href="#">Table 16</a> for values.
1:4	uint32	<b>FileAttributeValue</b> This is a fixed-length return value. See <a href="#">Table 12</a> .

**Table 12 — DfGetFileAttribute Returned Value Definition**

FileAttributeName	Definition
ClientZeroLengthOnly	<p>If the <i>FileAttributeReq ClientZeroLengthOnly</i> is set to one (1), then the returned <i>ClientZeroLengthOnly</i> indicates, if set to one (1), this file has been designated by the File Client to be preserved until the File Client explicitly sets the <i>ClientZeroLengthOnly</i> to zero (0). The file shall be opened exclusively if this attribute is set to one (1).</p>
ChangeIndicator	<p>If the <i>FileAttributeReq RequestCI</i> is set to one (1), the returned <i>ChangeIndicator</i> is generated by the File Host either at the time of the reception of the <i>DfGetFileAttribute</i> or at the time when the file was last changed. The File Client may compare the current value to a previously saved value to indicate if the file has changed since the last time the File Client read the file. The <i>ChangeIndicator</i> should be a 32-bit CRC.</p>
RequestMaxPoll	<p>If the <i>FileAttributeReq ReqMaxPoll</i> is set to one (1), the returned <i>RequestMaxPoll</i> is the maximum time, in milliseconds, allowed between reading the <a href="#">File Size Monitoring Sensor</a> or <i>DfRead</i> command before the data may either truncate or wrap, depending on the <i>File PDR FileCapabilities</i> settings.</p>

## 211 9.5 DfSetFileAttribute Command

---

- 212 The DfSetFileAttribute command, as described in [Table 13](#), is used by the File Client to set specific dynamic file attributes such as file preservation.
- 213 If the *FileIdentifier* requested by the File Client is invalid, the File Host shall return an INVALID\_FILE\_IDENTIFIER *CompletionCode*.
- 214 If the requested *FileIdentifier AttributeReq* is not supported by the File Host, then it shall return an INVALID\_DF\_ATTRIBUTE *CompletionCode*.
- 215 If more than one bit is set in *AttributeReq*, then the File Host shall return an INVALID\_DF\_ATTRIBUTE *CompletionCode*.
- 216 If the file specified by the *FileIdentifier* has a valid *FileDescriptor* by any File Client, then the File Host shall return the FILE\_OPEN *CompletionCode*.
- 217 If the SUCCESS *CompletionCode* is not returned, then no change is made to the DfSetFileAttribute.

218

**Table 13 — DfSetFileAttribute Command Format**

Byte	Type	Request Data
0:1	uint16	<b>FileIdentifier</b> This is the <i>FileIdentifier</i> returned in the <i>File PDR</i> for this file (directory).
2:3	bitfield16	<b>FileAttributeSet</b> 1b = Request setting specified attribute 0b = setting of attribute not requested [0] ClientZeroLengthOnly [1:15] Reserved (0)
4:7	uint32	<b>FileAttributeValue</b> This is a fixed-length value. See <a href="#">Table 14</a> .
Byte	Type	Response Data
0	enum8	<b>CompletionCode</b> Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIBUTE, INVALID_FILE_IDENTIFIER, FILE_OPEN } See <a href="#">Table 16</a> for values.

**Table 14 — DfSetFileAttribute SUCCESS Value Definition**

FileAttributeName	Definition
ClientZeroLengthOnly	<p>If <i>ClientZeroLengthOnly FileAttributeValue</i> is set to one (1), then the file has been designated by the File Client to be preserved until explicitly released by the File Client. Setting the <i>ClientZeroLengthOnly FileAttributeValue</i> to zero (0) allows the file to be deleted or updated by the File Host.</p> <p>The file shall be subsequently opened exclusively if this attribute is set to one.</p> <p>If the <i>ClientZeroLengthOnly FileAttributeValue</i> bit is set to (1), the File Host shall not update or add to the file after initial creation.</p> <p>The File Client is only allowed to change the <i>ClientZeroLengthOnly FileAttributeValue</i> if the file is not currently open.</p> <p>If the file is currently open, the File Host and the File Client shall not change the <i>ClientZeroLengthOnly</i> state and the File Host shall return the <i>FILE_OPEN CompletionCode</i>.</p> <p>The <i>ClientZeroLengthOnly</i> state shall be preserved across resets and power cycles for <i>non-volatile</i> files <a href="#">File PDR FileCapabilities DataVolatility</a>. The <i>ClientZeroLengthOnly</i> state for <i>volatile</i> files should be preserved across resets.</p>

## 220 9.6 DfHeartbeat Command

---

221 The DfHeartbeat command is a multiple function command, providing both an initialization / negotiation function and a simple flow control function for *SerialTxFIFO* type files.

222 The DfHeartbeat command, as described in [Table 15](#), enables:

- initialization to negotiate the maximum time interval allowed between the last DfOpen, DfRead, or DfHeartbeat command and when the File Host may optionally close the *FileDescriptor*,
- an indication to the File Host that the current *FileDescriptor* is still active (also known as keep alive), even with no periodic DfRead activity,
- the File Client or the File Host to request a shorter or longer maximum time interval as a flow control function.

223 The maximum time interval may be negotiated during any DfHeartbeat command invocation. The File Host is permitted to request a different *ResponderMaxInterval* when the file data is not retrieved at a rate to avoid an overflow or truncation condition. This method typically is used to inform the File Client when a *SerialTxFIFO* file is approaching capacity and needs a faster polling DfRead to avoid dropping data. The File Client may also request a different *RequesterMaxInterval*, but this is not the usual expected use case (flow) since the File Client can control the polling rate for the DfRead command with the invocation frequency and, for the DfFIFOSend command, the File Client can increase / decrease the DfFIFOSend response rate.

224 Upon successful completion of each invocation of this command, the lesser value of *RequesterMaxInterval* and *ResponderMaxInterval* is defined as the current *NegotiatedInterval*. The File Client shall issue DfHeartbeat or DfRead commands using the current *NegotiatedInterval* as the maximum period between DfOpen, DfRead, and DfHeartbeat commands.

225 If the *FileDescriptor* requested by the File Client is not valid or is no longer valid, the File Host shall return an `INVALID_FILE_DESCRIPTOR` *CompletionCode*.

### 226 9.6.1 Implicit File Close

227 If the File Host has not received a DfHeartbeat or a DfRead command within the current *NegotiatedInterval*, it may optionally do an implicit file close of the *FileDescriptor*. If the File Host has closed the *FileDescriptor*, then it shall return an `INVALID_FILE_DESCRIPTOR` *CompletionCode* on any uses of that *FileDescriptor* by the File Client.

228 If there is no current valid *NegotiatedInterval* and if the DfHeartbeat command is not sent within an implementation-specific amount of time, then the File Host may do an implicit file close.

229

**Table 15 — DfHeartbeat Command Format**

Byte	Type	Request Data
0:1	uint16	<b>FileDescriptor</b>
2:5	uint32	<b>RequesterMaxInterval</b> The requested maximum supported <i>NegotiatedInterval</i> in milliseconds
Byte	Type	Response Data
0	enum8	<b>CompletionCode</b> Possible values: { PLDM_BASE_CODES, INVALID_FILE_DESCRIPTOR } See <a href="#">Table 16</a> for values.
1:4	uint32	<b>ResponderMaxInterval</b> The maximum supported <i>NegotiatedInterval</i> in milliseconds from the responder

230 **9.7 Error Completion Codes**

231 PLDM completion codes for file transfer that are beyond the scope of PLDM\_BASE\_CODES in DSP0240 are defined in Table 16. The contexts in which these codes are used are also described in the table below.

232 **Table 16 — PLDM File Transfer Completion Codes**

Value	Name	Returned By	Usage Description
Various	PLDM_BASE_CODES	File Host & File Client	See the <a href="#">DSP0240 PLDM Base Specification</a> .
0x80	INVALID_FILE_DESCRIPTOR	File Host & File Client	Invalid <i>FileDescriptor</i> was provided to one of the following commands: <a href="#">DfRead</a> , <a href="#">DfClose</a> , <a href="#">DfHeartbeat</a> .
0x81	INVALID_DF_ATTRIBUTE	File Host	Invalid attribute or combinations of attributes was provided to one of the following commands: <a href="#">DfOpen</a> , <a href="#">DfGetFileAttribute</a> , <a href="#">DfSetFileAttribute</a> .
0x82	ZEROLENGTH_NOT_ALLOWED	File Host	<a href="#">DfClose</a> command <i>ZeroLength</i> option set to one (1) of this file is not allowed. See <a href="#">DfClose</a> .
0x83	EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED	File Host	Attempted to use <a href="#">DfClose</a> command <i>ZeroLength</i> option set to one (1) without proper ownership. See <a href="#">DfClose</a> .
0x84	EXCLUSIVE_OWNERSHIP_NOT_ALLOWED	File Host	Requested file is not allowed to be opened exclusively. See <a href="#">DfOpen</a> .
0x85	EXCLUSIVE_OWNERSHIP_NOT_AVAILABLE	File Host	Requested file temporarily cannot be opened exclusively.
0x86	INVALID_FILE_IDENTIFIER	File Host	Invalid <i>FileIdentifier</i> was provided to one of the following commands: <a href="#">DfOpen</a> , <a href="#">DfGetFileAttribute</a> , <a href="#">DfSetFileAttribute</a> .
0x87	DFOPEN_DIR_NOT_ALLOWED	File Host	Opening a directory is not allowed. See <a href="#">DfOpen</a> .
0x88	MAX_NUM_FDS_EXCEEDED	File Host	A File Host has run out of <i>FileDescriptors</i> either for this file or overall. See <a href="#">DfOpen</a> .
0x89	FILE_OPEN	File Host	Attempted to change a file attribute on a currently opened file. See <a href="#">DfSetFileAttribute</a> .
0x8A	UNABLE_TO_OPEN_FILE	File Host	The File Host is temporarily unable to open a file. See <a href="#">DfOpen</a> .
0x8B-0xFF	Reserved	Reserved	



## 233 9.8 DfRead (DSP0240 MultipartReceive)

---

234 The DfRead command is a PLDM for File Transfer (type) specific implementation of the [DSP0240 PLDM Base Specification Multipart Transfer Commands](#), and specifically the MultipartReceive command. The [DSP0240 MultipartReceive](#) command allows the File Client to initiate a data transfer command (e.g., DfRead) from the File Host. The *Multipart Transfer Commands* allow a PLDM specification to define specific context to the command parameters, which allows this definition of the DfRead command to map values to the *Multipart Transfer Commands*.

235 A DfRead command is used to read one (1) multipart section using the [MultipartReceive](#) command. The size of the DfRead command (multipart section size) is determined by the File Client and is based on how much of the file the File Client wants to read and the maximum amount of data it wants to re-receive in case of an error and subsequent retransmission by the File Host.

236 To read a file sequentially using multiple DfRead commands, the File Client computes the *FileOffset* for the next DfRead command by using the previous *FileOffset* and adding the previous returned *DataLengthBytes*.

237 For this specification, the maximum [Multipart Transfer Commands Transfer Block](#) size is the current *File Size* value. The File Client may read all or part of the file represented by the *File Size* using the appropriate number of DfRead commands with appropriate *FileOffset*. The *Transfer Block* is only known and used by the File Client to know how many DfRead commands (multipart sections) it needs to issue.

238 The File Host shall not invalidate the *FileDescriptor* or close the file if an error completion code is sent or if the File Client sets the *TransferOperation* parameter to XFER\_ABORT. The File Client and File Host shall use the MultipartReceive command as specified in [DSP0240](#) with the mappings defined in [Table 19](#).

239 The File Host, as the MultipartReceive command responder, shall respond with a *CompletionCode* set to ERROR\_INVALID\_TRANSFER\_CONTEXT if the File Client MultipartReceive command request provides an invalid *FileDescriptor*.

240 There is no defined behavior for the File Client after it issues the XFER\_ABORT, and is out of scope of this specification.

241 This specification requires that PLDM for File Transfer *PLDMType* seven (7) is specified in the NegotiateTransferParameters command fields *RequesterProtocolSupport* and *ResponderProtocolSupport*.

242 A DfRead command within the *NegotiatedInterval* is equivalent to executing the DfHeartbeat command.

### 243 9.8.1 Serial FIFO type file characteristics

244 Files classified as a *SerialTxFIFO* have specific characteristics, similar to an endpoint streaming data of a Universal Asynchronous Receiver-Transmitter (UART) device. The following requirements apply:

- Seeking is not supported (MultipartReceive *Requested SectionOffset* shall be set to zero).
- Single part per section (TransferOperation shall not be set to XFER\_NEXT\_PART)
- Single section (no piggybacking multiple sections as the offset is always zero)
- The File Host shall move the *SerialTxFIFO* read pointer when the File Client sets the MultipartReceive *TransferOperation* field to XFER\_COMPLETE.

- The File Client shall issue the MultipartReceive to read a *SerialTxFIFO* type file until the data length is less than the negotiated part size, indicating that all the available data has been transferred.
- See [DfOpen SerialTxFIFO requirements](#) for additional DfOpenAttribute requirements.

245 The DfRead command is implemented using the *Multipart Transfer Commands*, as [Table 6](#) describes.

246 **9.8.2 DfRead command details**

247 The DfRead command maps to the MultipartReceive command as described in [Table 19](#). The DfRead command Request Data fields not specified in this table or by the following requirements are set to the MultipartReceive command defaults.

248 DfRead File Client request requirements are detailed in [Table 17](#):

249 **Table 17 — DfRead File Client Request Requirements**

<i>MultipartReceive TransferOperation</i>	All File Types	Additional Requirements when <i>DfOpenRegFIFO</i> is set to one (1).
XFER_FIRST_PART	<p>The <i>FileOffset</i> may be zero (0) (beginning of file) or a nonzero value representing the file offset.</p> <p>The initial <i>DataTransferHandle</i> shall be zero (0).</p>	<p>The <i>FileOffset</i> shall be set to zero (0).</p> <p>The <i>RequestedSectionLengthBytes</i> shall be equal to or less than the <i>Negotiated Transfer Part Size</i> from the most recent successfully completed <i>NegotiateTransferParameters</i> command.</p>
XFER_NEXT_PART		Not supported
XFER_COMPLETE		<i>RequestedSectionOffset</i> shall be zero (0).

250 DfRead File Host response requirements are detailed in [Table 18](#):

251 **Table 18 — DfRead File Host Response Requirements**

<i>MultipartReceive TransferOperation</i>	All File Types	Additional <i>SerialTxFIFO</i> File Type Requirements
XFER_FIRST_PART		The File Host shall respond with a <i>TransferFlag</i> equal to START_AND_END, as required by <a href="#">DSP0240</a> when <i>DataLengthBytes</i> is less than or equal to the <i>Negotiated Transfer Part Size</i> .
XFER_COMPLETE		The File Host shall move the read pointer ahead by the number of bytes successfully transferred.  The data is not retained by the File Host.

252

**Table 19 — DfRead Command to MultipartReceive Command Mapping Format**

Byte	Type	DfRead Request Data	MultipartReceive Request Data
0	uint8	0x07	PLDMType
1	enum8		TransferOperation
2:5	uint32	<b>FileDescriptor</b>	TransferContext
6:9	uint32	Initially 0	DataTransferHandle
10:13	uint32	<b>FileOffset</b>	RequestedSectionOffset
14:17	uint32		RequestedSectionLengthBytes

Byte	Type	DfRead Response Data	MultipartReceive Response Data
0	enum8		CompletionCode (MultipartReceive command)
1	enum8		TransferFlag
2:5	uint32		NextDataTransferHandle
6:9	uint32		DataLengthBytes
10:N+9	uint8[N]		Data
N+10:N+13	uint32		DataIntegrityChecksum

## 253 9.9 DfFIFOSend (DSP0240 MultipartSend)

---

- 254 The DfFIFOSend command is used exclusively for a *File PDR* with *FileCapabilities DataType* set to one (1) (*Streaming FIFO*) and *Pushed* set to one (1) and is opened by the File Client with the following DfOpen command DfOpenAttribute set:
- *DfOpenRegFIFO* — *Streaming FIFO* (serial FIFO) (1)
  - *DfOpenPolledPushed* — *Pushed* (1)
- 255 The DfFIFOSend command is equated to a PLDM for File Transfer (*PLDMType* seven (7)) specific implementation of the [DSP0240 PLDM Base Specification Multipart Transfer Commands](#), and specifically the MultipartSend command. The MultipartSend command allows the File Host to initiate a data transfer (e.g., DfFIFOSend) to the File Client. The *Multipart Transfer Commands* allow a PLDM specification to define specific context to the command parameters, which allows this definition of the DfFIFOSend command to map values to the MultipartSend command.
- 256 The File Client shall be prepared to respond successfully to a DfFIFOSend command request after the File Host has successfully responded to a DfOpen command.
- 257 The File Host asynchronously, without prompting, when file data is placed in the *SerialTxFIFO* file, initiate a data transfer from the File Host to the File Client. The File Host then waits for the reception acknowledgment to be received. The transfer semantics are defined in [DSP0240 PLDM Base Specification MultipartSend](#) command but using the PLDM for File Transfer field mappings in [Table 20](#).
- 258 Similar to the DfRead command, upon receiving the multipart XFER\_ABORT operation from a File Client in response to MultipartSend command, a File Host shall discard the entire transfer and the *DataTransferHandle* is invalidated. The file shall not be closed and the *FileDescriptor* shall remain valid.
- 259 There is no defined behavior for the File Client after the issuance of the XFER\_ABORT *TransferFlag*, and is out of scope for this specification.
- 260 The File Host shall make *SectionLengthBytes* equal to *DataLengthBytes* and equal to or less than the *Negotiated Transfer Part Size* from the most recent successfully completed *NegotiateTransferParameters* command. Upon reception of the *NextTransferOperation* with XFER\_COMPLETE, the read pointer shall be moved ahead by the number of bytes successfully transferred and this data section is no longer re-transmittable.

261

**Table 20 — DfFIFOSend to MultipartSend Command Mapping Format**

Byte	Type	DfFIFOSend Request Data	MultipartSend Data
0	uint8	0x07	PLDMType
1	enum8	START_AND_END	TransferFlag
2:5	uint32	<b>FileDescriptor</b>	TransferContext
6:9	uint32	0	DataTransferHandle
10:13	uint32	0 (DSP0240)	NextDataTransferHandle
14:17	uint32	0 (DSP0242)	SectionOffset
18:21	uint32		SectionLengthBytes
22:25	uint32		DataLengthBytes
26:N+25	uint8[N]		Data
N+26:N+29	uint32		DataIntegrityChecksum

Byte	Type	DfFIFOSend Response Data	MultipartSend Response Data
0	enum8		CompletionCode (MultipartSend command)

262

## 10 ANNEX A (informative) Change Log

---

Version	Date	Description
1.0.0	2024-07-29	Initial Release

263 **11 ANNEX B (informative) Sensor Threshold Event Examples**

264 [Table 21](#) lists examples of:

- *File Size Monitoring Sensor* thresholds, and
- possible events from both the *File Size Monitoring Sensor* and *Device File State Sensor*.

265 All File Size Monitoring Sensor Low thresholds are zero (0).

266 **Table 21 — Sensor Thresholds and Event Examples**

<i>FileClassification</i>	FMS	PFS CFS	WH	CH	FH	DFSS	FSMS Event	Note
SerialTxFIFO	256	0 1	128	230	256	FiU		
SerialTxFIFO	256	1 1	128	230	256	FhNC		
SerialTxFIFO	256	1 1	128	230	256	FhNC		
SerialTxFIFO	256	1 200	128	230	256	FhNC	WH	
SerialTxFIFO	256	256 256	128	230	256	Fdwh	FH	
CrashDumpFile		none 1024					PDR RU	Very expensive
CrashDumpFile	1024	0 1024	512	920	1024	FiaMS	CH	
CrashDumpFile	1024	1024 0	512	920	1024	FiU		DfClose ZeroLength=1
FRUDataFile OtherFile	1024	1024 1024						existing <i>Static File</i>
FRUDataFile OtherFile	1024	1024 1024				FiU		existing <i>Fixed Length File</i> with update
BootLog DiagnosticLog SecurityLog OtherLog	1024	100 101	512	920	1024	FiU		Crossed WH



<i>FileClassification</i>	FMS	PFS CFS	WH	CH	FH	DFSS	FSMS Event	Note
BootLog DiagnosticLog SecurityLog OtherLog	1024	101 513	512	920	1024	FiU	WH	
<i>FileClassification</i>	FMS	PFS CFS	WH	CH	FH	DFSS	FSMS Event	Note
Blank Row	1111	2222 3333	5555	7777	9999	DFSS	FSMS	Note

267 FMS = *FileMaximumSize*, PFS = *Prior File Size*, CFS = *Current File Size*, WH = *WarningHigh*, WL = *WarningLow*, CH = *CriticalHigh*, CL = *CriticalLow*, FH = *FatalHigh*, FL = *FatalLow*,

268 DFSS = *Device File State Sensor*, FiU = *File is Updated*, FDhW = *File Data has Wrapped*, FDhT = *File Data has Truncated*, FiaMS = *File is at Maximum Size*, FhNC = *File has Not Changed*

269 FSMS Event = *File Size Monitoring Sensor Event*

270 PDR RU = PLDM PDR Repository Update to show a new file

## 271 **12 ANNEX C (informative) File PDR FileClassification FileCapabilities Examples**

---

272 [Table 22](#) lists examples of *FileClassification* and *FileCapabilities* from the *File PDR*, with a compatible set of DfOpen command attributes and description of the commands that are used to access that file.

**Table 22 — FileClassification FileCapabilities DfOpen command attributes examples**

<i>FileClassification</i>	<i>FileCapabilities</i>	<i>DfOpen Attributes</i>	<i>Description</i>
SerialTxFIFO	ExReadOpen=0 FileTrunc=1 ( <i>Truncate</i> ) DataType=1 ( <i>FIFO</i> ) Polled=1 ( <i>Polled</i> ) Pushed=1 ( <i>Pushed</i> ) DataVolatility=0 ( <i>Volatile</i> ) FileModify=0 ( <i>Append</i> ) FcZeroLengthPermitted=0 FcWritesPermitted=0	DfOpenReadWrite= <i>Read</i> DfOpenRegFIFO= <i>FIFO</i> DfOpenPolledPushed= <i>Polled</i>	File supports polled or pushed read SerialTxFIFO access.  The DfOpen command is for Polled streaming read of the SerialTxFIFO using the DfRead / Multipart Receive command.
SerialTxFIFO	ExReadOpen=0 FileTrunc=1 ( <i>Truncate</i> ) DataType=1 ( <i>FIFO</i> ) Polled=1 ( <i>Polled</i> ) Pushed=1 ( <i>Pushed</i> ) DataVolatility=0 ( <i>Volatile</i> ) FileModify=0 ( <i>Append</i> ) FcZeroLengthPermitted=0 FcWritesPermitted=0	DfOpenReadWrite= <i>Read</i> DfOpenRegFIFO= <i>FIFO</i> DfOpenPolledPushed= <i>Pushed</i>	File supports polled or pushed read SerialTxFIFO access.  The DfOpen command is for pushed streaming reads of the SerialTxFIFO using the DfFIFOSend / Multipart Send command.
CrashDumpFile	ExReadOpen=1 FileTrunc=1 ( <i>Truncate</i> ) DataType=0 ( <i>Regular</i> ) Polled=1 ( <i>Polled</i> ) Pushed=0 DataVolatility=1 ( <i>Non-volatile</i> ) FileModify=0 ( <i>Append</i> ) FcZeroLengthPermitted=1 FcWritesPermitted=0	DfOpenReadWrite= <i>Read</i> DfOpenExclusive= <i>Non-exclusive</i> DfOpenRegFIFO= <i>Regular</i> DfOpenPolledPushed= <i>Polled</i>	File supports <i>Exclusive</i> or <i>Non-exclusive</i> read access.  The DfOpen command is for <i>Regular</i> Non-exclusive reads using the DfRead / Multipart Receive command.
CrashDumpFile	ExReadOpen=1 FileTrunc=1 ( <i>Truncate</i> ) DataType=0 ( <i>Regular</i> ) Polled=1 ( <i>Polled</i> ) Pushed=0 DataVolatility=1 ( <i>Non-Volatile</i> ) FileModify=0 ( <i>Append</i> ) FcZeroLengthPermitted=1 FcWritesPermitted=0	DfOpenReadWrite= <i>Read</i> DfOpenExclusive= <i>Exclusive</i> DfOpenRegFIFO= <i>Regular</i> DfOpenPolledPushed= <i>Polled</i>	File supports <i>Exclusive</i> or <i>Non-exclusive</i> read access.  The DfOpen command is for exclusive reads using the DfRead / Multipart Receive command with the option to use DfClose command with the <i>ZeroLength=1</i> option.

<i>FileClassification</i>	<i>FileCapabilities</i>	<i>DfOpen Attributes</i>	<i>Description</i>
BootLog DiagnosticLog SecurityLog FRUDataFile OtherLog OtherFile	ExReadOpen=0 FileTrunc=0 ( <i>Truncate</i> ) DataType=0 ( <i>Regular</i> ) Polled=1 ( <i>Polled</i> ) Pushed=0 DataVolatility=1 ( <i>Non-volatile</i> ) FileModify=0 ( <i>Append</i> ) FcZeroLengthPermitted=0 FcWritesPermitted=0	DfOpenReadWrite= <i>Read</i> DfOpenExclusive= <i>Non-exclusive</i> DfOpenRegFIFO= <i>Regular</i> DfOpenPolledPushed= <i>Polled</i>	File supports only non-exclusive read access.  The DfOpen command is for non-exclusive reads using the DfRead / Multipart Receive command with no option to use DfClose command with the <i>ZeroLength=1</i> option.

## 274 **13 ANNEX D (informative) PLDM for File Transfer Examples**

---

275 This informative section describes typical flows involving File Transfer commands.

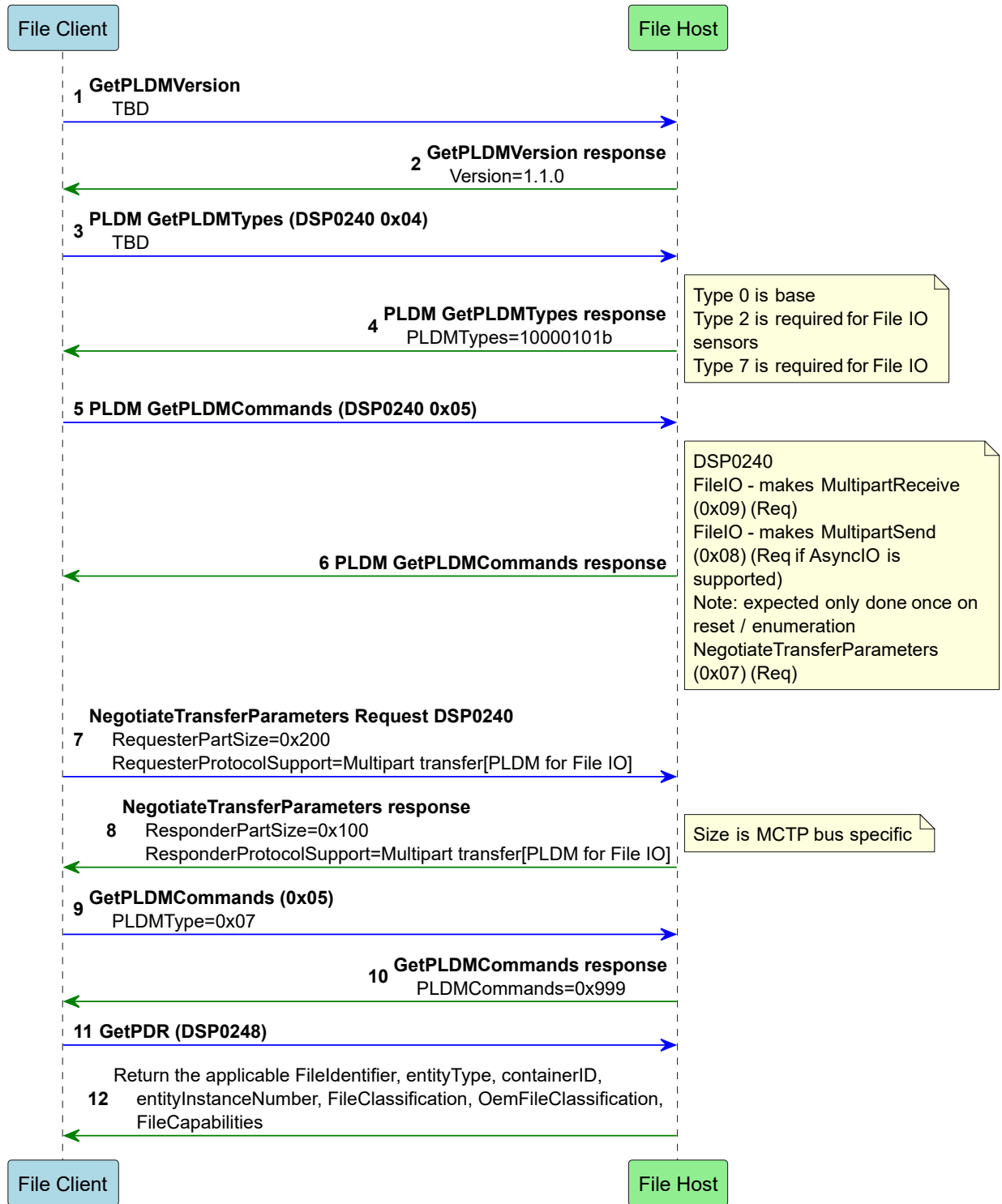
1. [Initialization example](#)
2. [Regular log file read](#)
3. [Polled Serial log read](#)
4. [Pushed Serial log read](#)

### 276 **13.1 PLDM for File Transfer initialization example**

---

277 Figure 4 shows an example of the PLDM for File Transfer initialization sequence.

Example PLDM for File Transfer Initialization



279

**Figure 4 — PLDM for File Transfer Initialization Example**



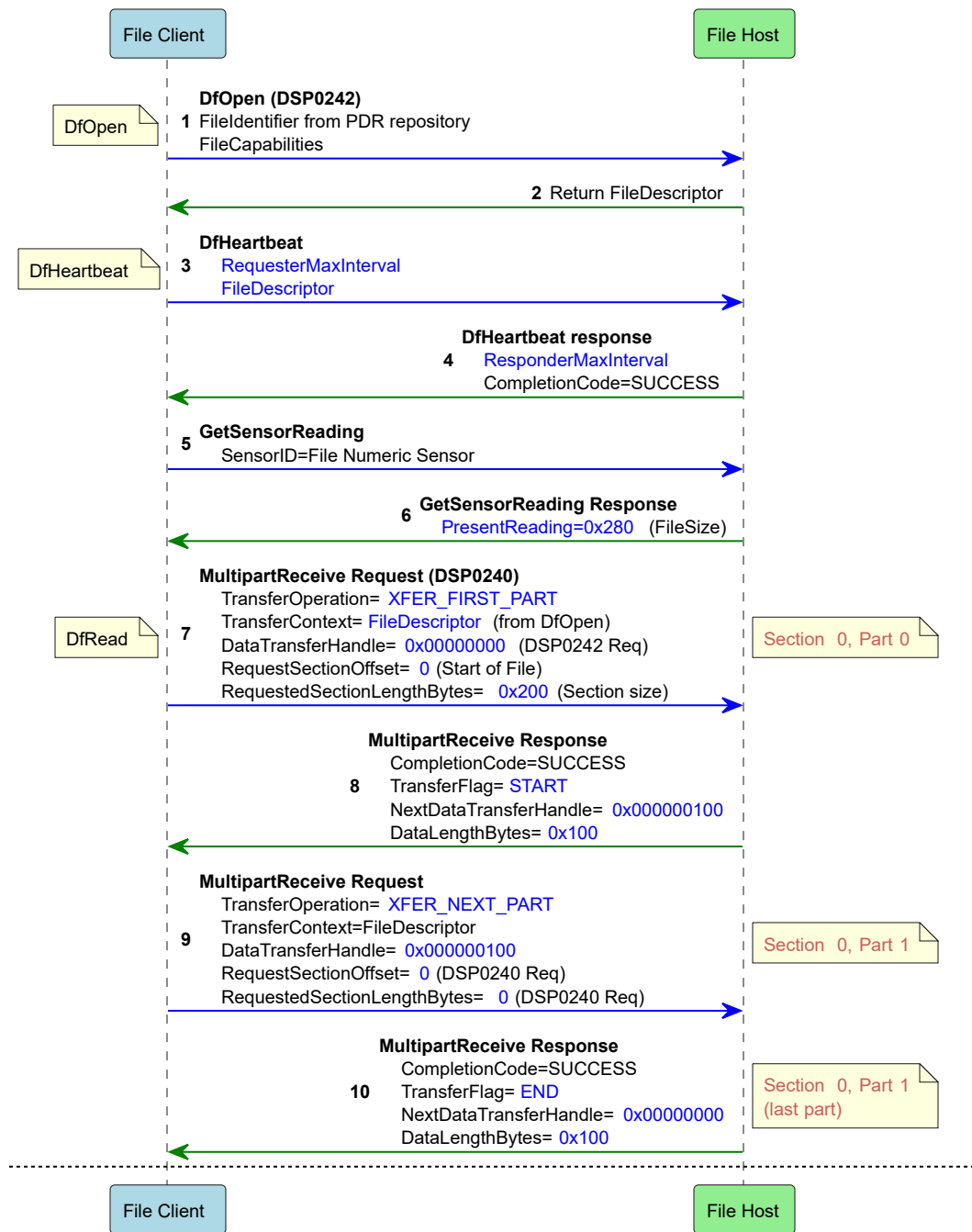
## 280 **13.2 Regular log file read**

---

281 Figures 5, 6, 7, and 8 show an example of a *Regular* log file read. The part size of 0x100 (256) bytes is a result of a previously executed NegotiateTransferParameters command. As defined in the [DSP0240 PLDM Base Specification](#), the NextDataTransferHandle returned from the FileHost and required to be provided by the File Client on subsequent Parts is totally defined by the File Host and the values are opaque to the File Client.

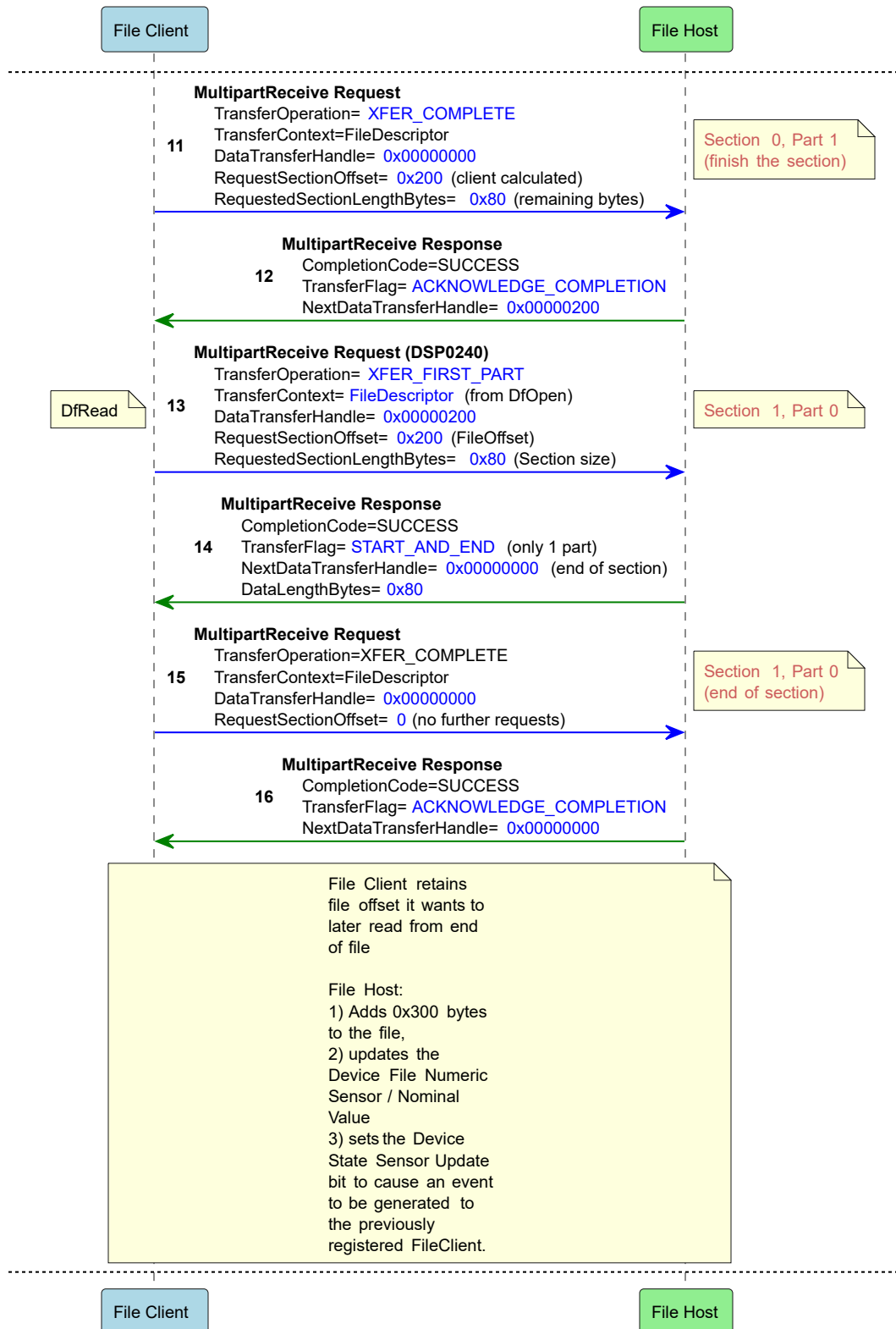
- [Figure 5](#) shows an example of a logical block incrementing NextDataTransferHandle.
- [Figure 6](#) shows an example of a sequential incrementing NextDataTransferHandle.
- [Figure 7](#) shows an example of Multipart transfer with a TransferFlag = Middle.
- [Figure 8](#) shows an example of reading a file at the end of file mark.

282



283

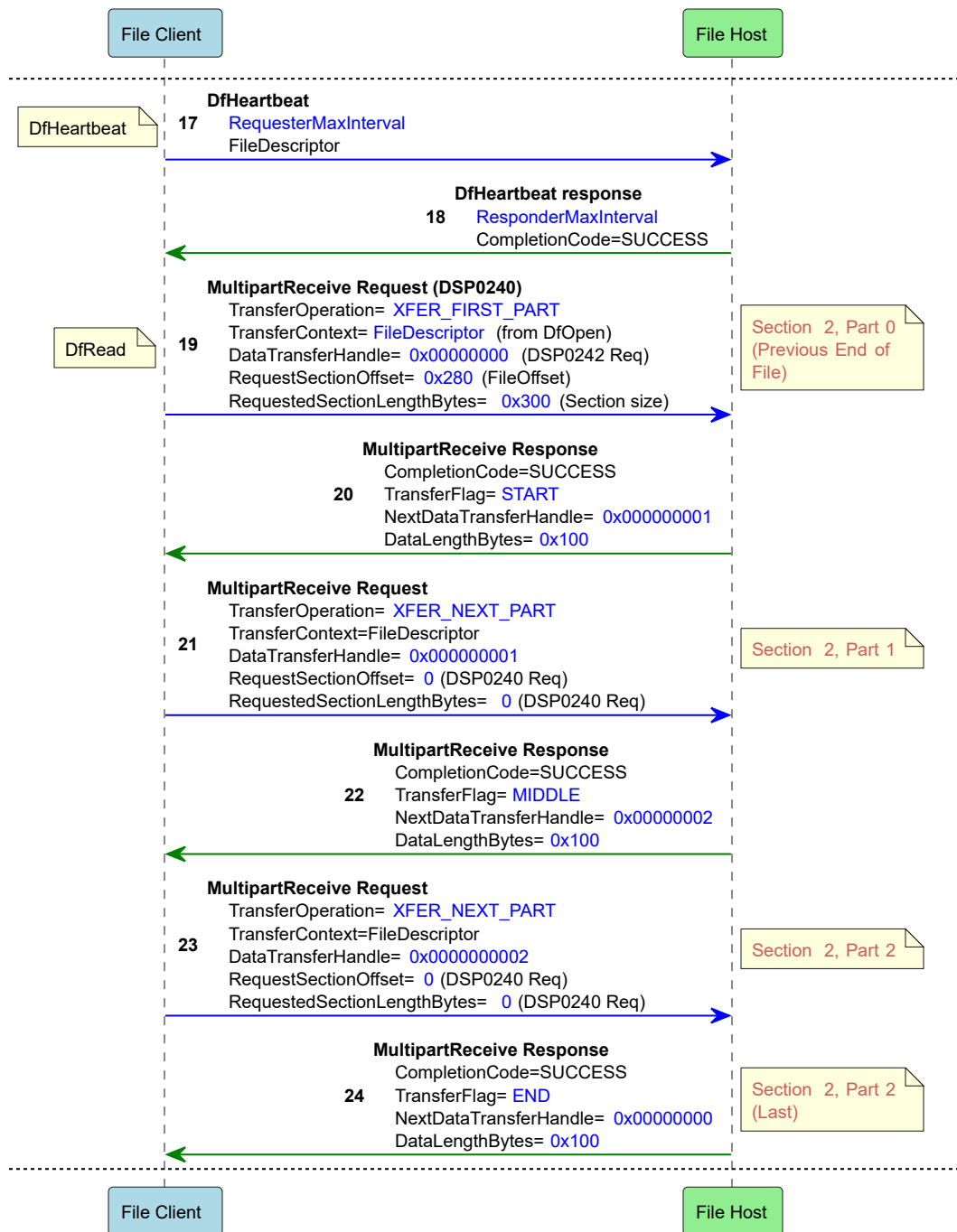
Figure 5 — Regular Log File Read Example - Page 1



285

**Figure 6 — *Regular* Log File Read Example - Page 2**

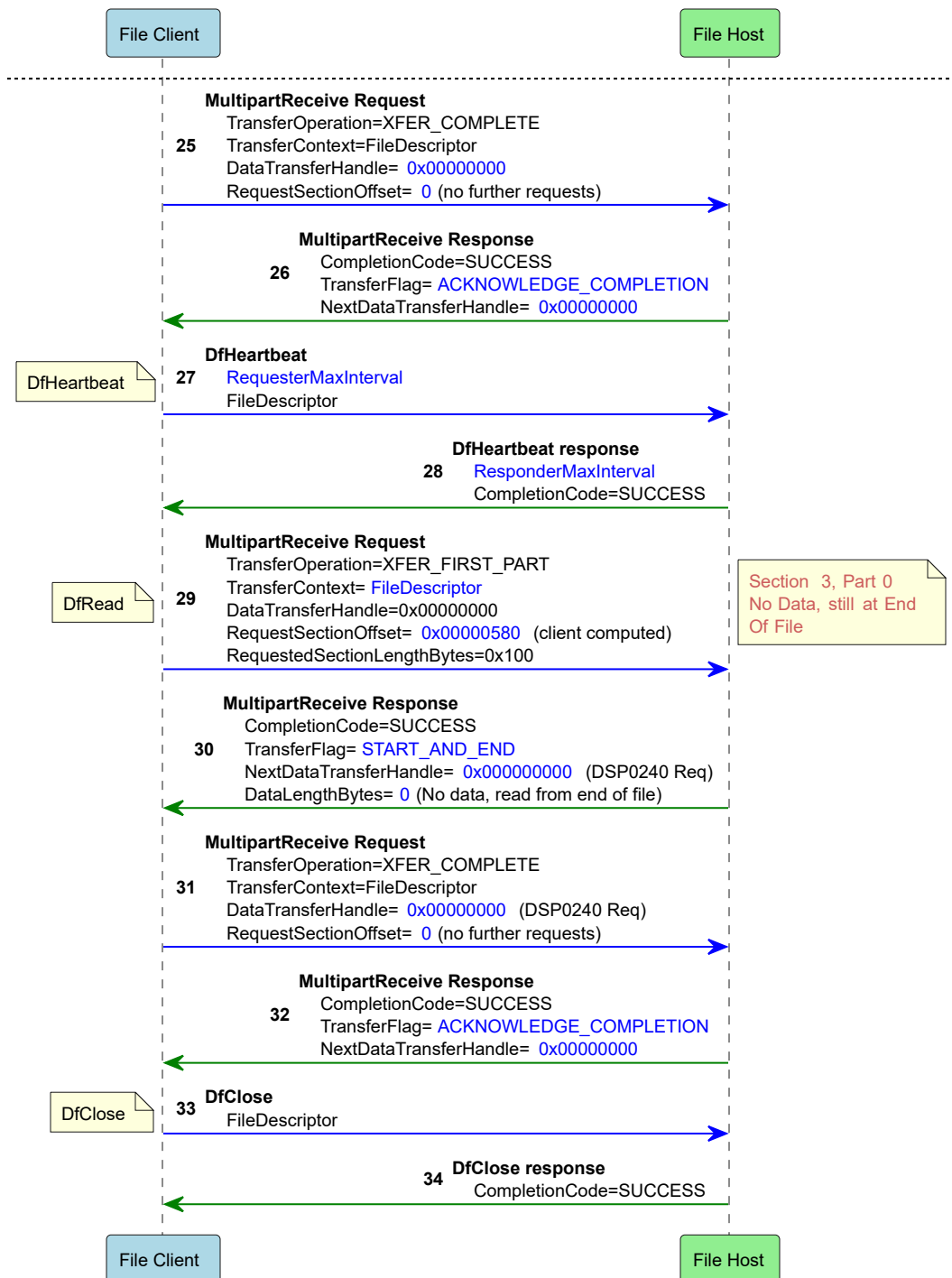
286



287

Figure 7 — Regular Log File Read Example - Page 3

288



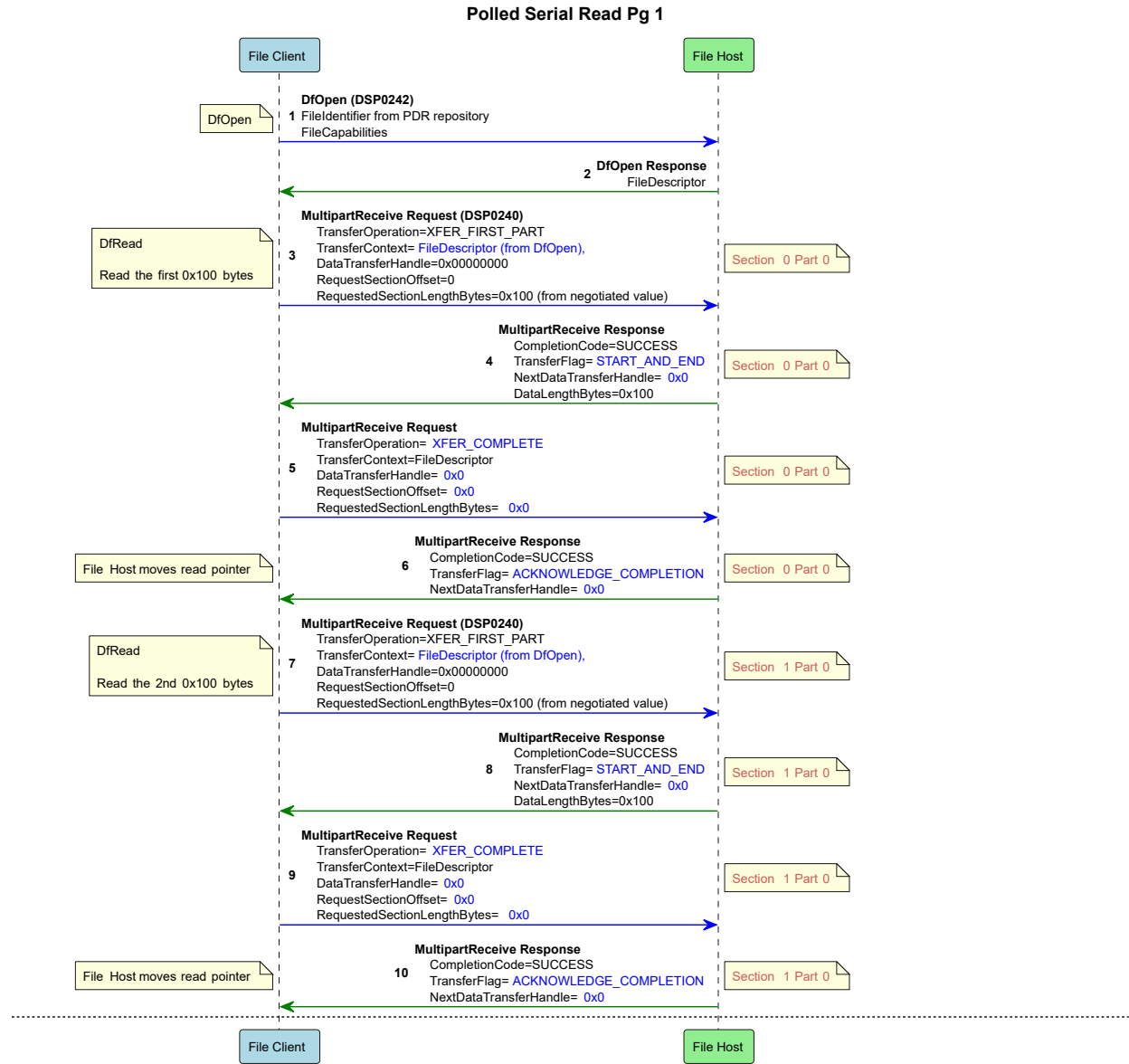
289

Figure 8 — Regular Log File Read Example - Page 4

290 **13.3 Polled Serial Log Read**

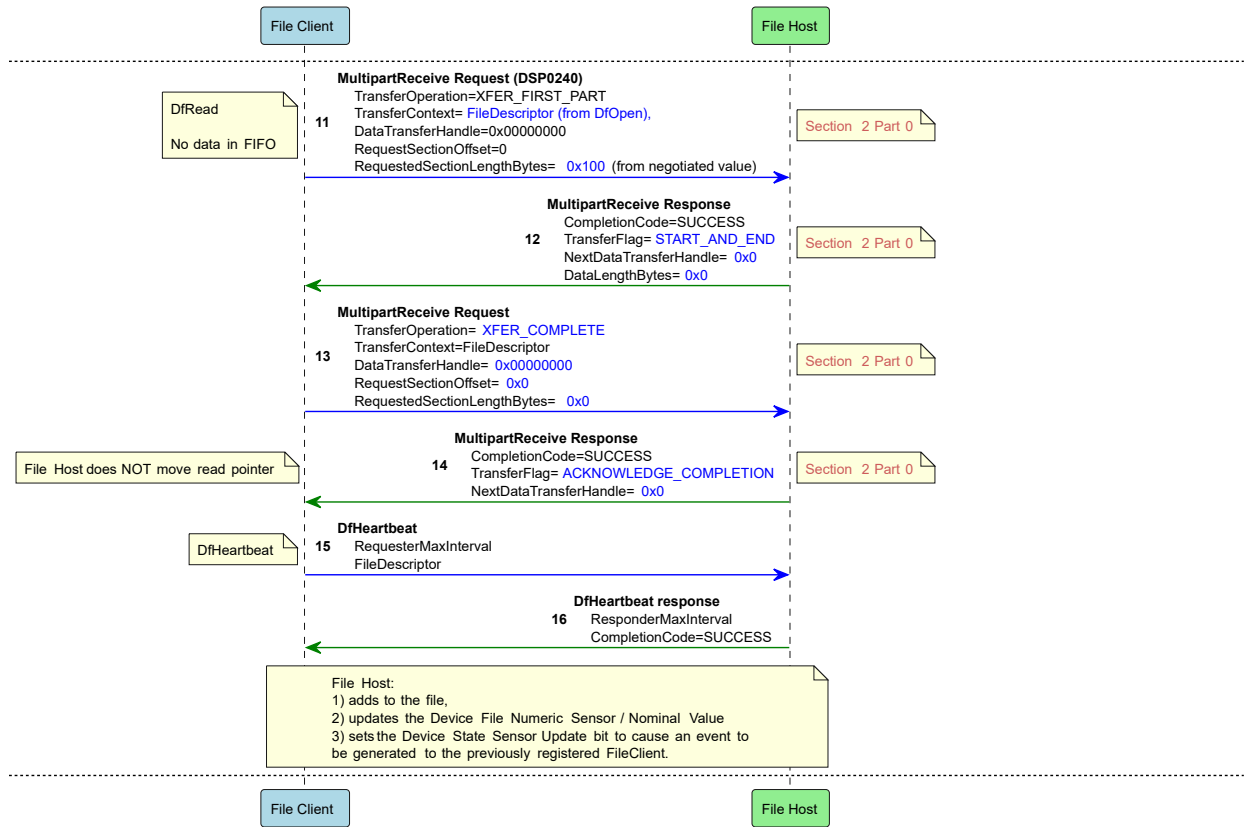
291 Figures 9, 10, and 11 show an example of a *SerialTxFIFO* log read.

292



293 **Figure 9 — Polled Serial Read Example - Page 1**

294

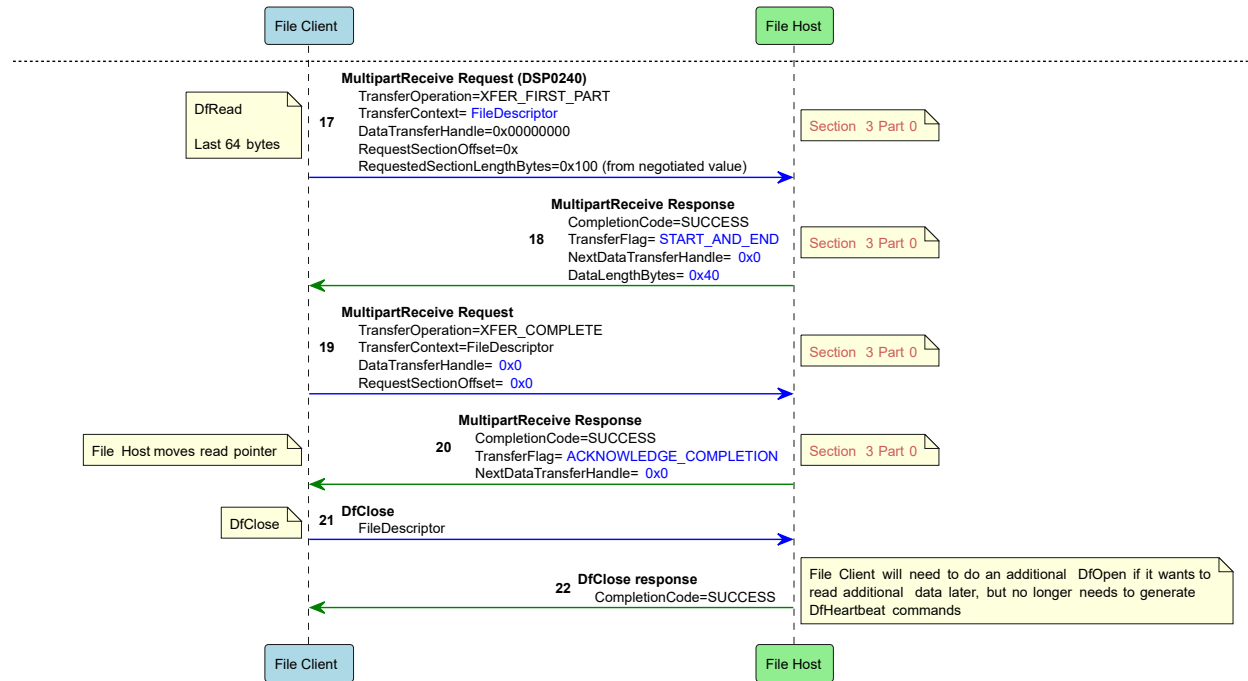


295

Figure 10 — Polled Serial Read Example - Page 2



296



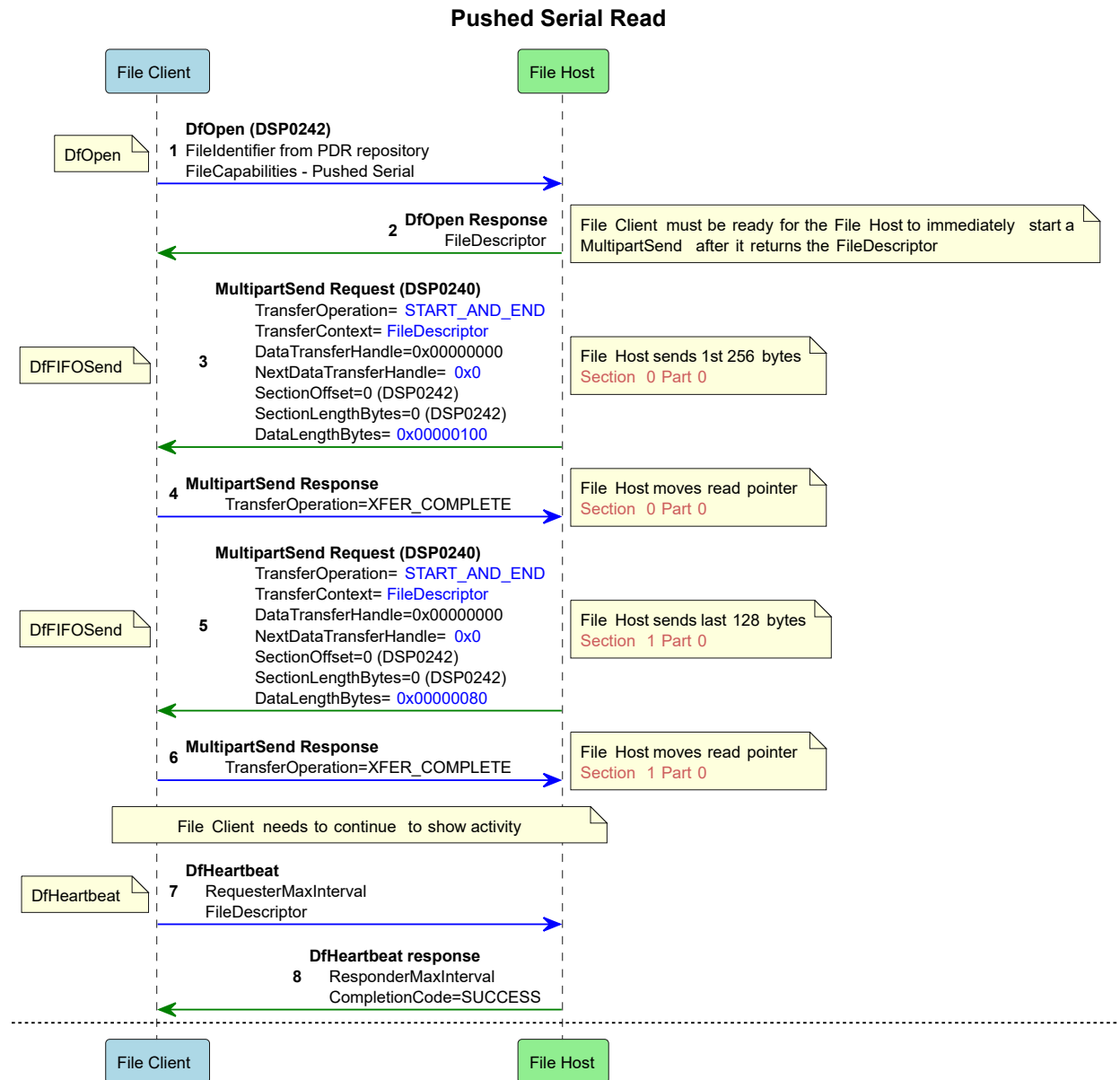
297

Figure 11 — Polled Serial Read Example - Page 3

298 **13.4 Pushed Serial Log Read**

299 Figures 12 and 13 show an example of a *Pushed SerialTxFIFO* log read.

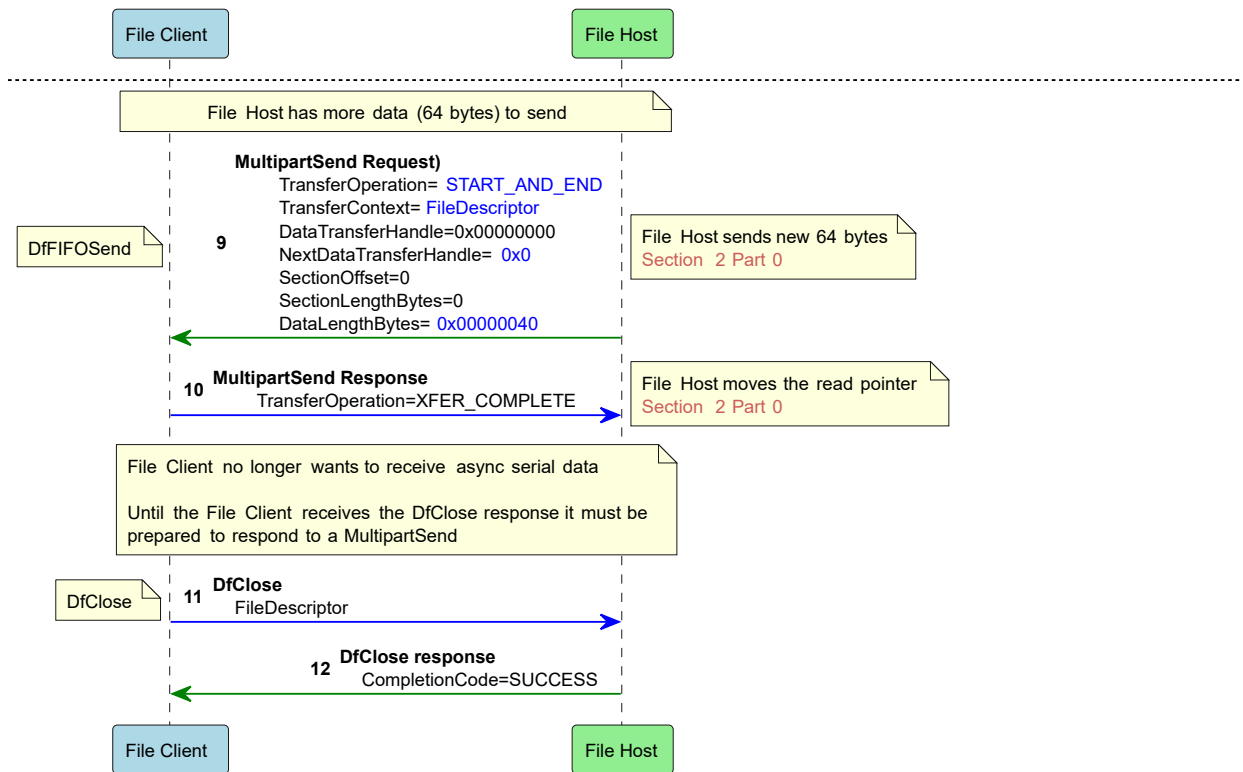
300



301

Figure 12 — Pushed SerialTxFIFO Log Read Example - Page 1

302



303

Figure 13 — Pushed SerialTx FIFO Log Read Example - Page 2