Platform Level Data Model (PLDM) for File Transfer Specification

Version: 1.0.1

Document Identifier: DSP0242

Date: 2026-01-07

Version History: https://www.dmtf.org/dsp/DSP0242

Supersedes: 1.0.0

Document Class: Normative

Document Status: Published

Document Language: en-US

10 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 for uses consistent with this purpose, 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.

11 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 products, protocols, or testing procedures. DMTF shall have no liability to any party implementing such standards, 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patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a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.

12 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.

13 IETF® is a registered trademark of IETF Trust. ISO® is a registered trademark of the International Organization for Standardization (ISO).

14 This document's normative language is English. Translation into other languages is permitted.

CONTENTS

15 # 1 Foreword

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

17 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.

18 ## 1.1 Acknowledgments

19 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

2 Introduction

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.

2.1 Document conventions

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

24

3 Scope

25 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

26 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

27 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.

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

29 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

4 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.

DMTF DSP0240 *Platform Level Data Model (PLDM) Base Specification* 1.2
https://www.dmtf.org/standards/published_documents/DSP0240_1.2.pdf

DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes* 1.4
https://www.dmtf.org/standards/published_documents/DSP0245_1.4.pdf

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

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

DMTF DSP1001, *Management Profile Specification Usage Guide* 1.1
https://www.dmtf.org/standards/published_documents/DSP1001_1.1.pdf

DMTF DSP4014, *DMTF Process for Working Bodies* 2.13
https://www.dmtf.org/sites/default/files/standards/documents/DSP4014_2.13.pdf

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

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

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

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

5 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.

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.

File Client

A PLDM Terminus that can receive files from a File Host

File Host

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

File PDR

File Descriptor Platform Descriptor Record (PDR) as defined in DSP0248 PLDM for Platform Monitoring and Control Specification

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.

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.

NegotiatedInterval

The maximum negotiated time interval in milliseconds to be used between commands issued by the File Client. See DfHeartbeat for requirements.

6 Symbols and abbreviated terms

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.

EAR

Entity Association PDR as defined in DSP0248 PLDM for Platform Monitoring and Control

FIFO

First in, first out

IANA

Internet Assigned Numbers Authority

OEM

Original Equipment Manufacturer

PDR

Platform Descriptor Record as defined in DSP0248 PLDM for Platform Monitoring and Control Specification

74

7 PLDM for File Transfer version

75 The version of this Platform Level Data Model (PLDM) for File Transfer Specification shall be 1.0.1 (major version number 1, minor version number 0, update version number 1, and no alpha version). In response to the GetPLDMVersion command described in DSP0240, the reported version for Type 7 (PLDM for File Transfer, this specification) shall be encoded as 0xF1F0F100.

76 # 8 PLDM for File Transfer Concepts

77 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.

78 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.

79 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.

80 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.

81 **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.

82 ## 8.1 File Metadata

83 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.

8.2 File Transfer

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:

- ***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
- ***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

8.3 File Discovery, Hierarchy and Identity Semantics

This section describes the terminology and semantics used by this specification as they relate to the ISO C Language FILE library semantics.

8.3.1 Semantics

- ***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.
- ***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

93 • ***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.

94 • ***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.

95 8.3.2 File Types and Classification

96 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.

97 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.

98 **NOTE:** Table 1 has some examples of file (data) classifications.

99 **Table 1 — *FileClassification* Examples**

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

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

100

101 8.3.3 File and Directory Discovery

102 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.

103 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, if it is the event receiver, 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.

104 8.3.4 File System Hierarchy Discovery

105 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.

106 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.

107 8.4 RequestMaxPoll Usage

108 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.

8.5 RequestCI Usage

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.

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.

8.6 DfOpenExclusive Usage

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.

8.7 File Zero Length

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.

8.7.1 ClientZeroLengthOnly Usage

> NOTE
>
> 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.
>
> 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:
>
> 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).

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.
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.
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*
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*
6. The File Client proceeds with a DfOpen with exclusivity and DfRead command sequence to retrieve the *CrashDumpFile*.
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.
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.

120 ## 8.8 DSP0248 PLDM for Platform Monitoring and Control Specification Relationship

121 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.

122 ### 8.8.1 The File Descriptor Data Model

123 *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*.

8.8.2 Required File Sensors

Table 2 describes the different file characteristics.

Table 2 — File Characteristic Definitions

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

NOTE: Table 3 describes the sensor usage for different file characteristics.

Table 3 — Sensor Usage for Different File Characteristics

File Characteristic	Usage
Static File	The file is always at maximum size, so having a *Device File State Sensor File is at Maximum Size State* is meaningless. File is a static size, so any *File Size Monitoring Sensor* thresholds are meaningless.
Fixed Length File	The *Device File State Sensor* is used to indicate and/or notify of a file update.
Regular File	The *FatalHigh* threshold is used to allow overflow detection. *Device File State Sensor* is used to indicate and/or notify of a file update.
SerialTxFIFO	The *File Size Monitoring Sensor WarningHigh* threshold allows the File Client to be notified it is not retrieving / accepting data faster than the data being written to the FIFO. The *File Size Monitoring Sensor FatalHigh* threshold allows the File Client to be notified that data has overflowed. The *Device File State Sensor* is used to indicate and/or notify of an overflow condition.

Table 4 describes the sensors and thresholds required for each file characteristic.

130 **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
Static File	Should	Should not	Should	Should not support *File is at Maximum File State*
Fixed Length File	Should	Should not	Shall	Should not support *File is at Maximum File State*
Regular File	Shall	Should	Shall	Should support all defined states
SerialTxFIFO	Shall	Shall	Shall	Should not support *File is Updated* or *File has Not Changed*

131 ### 8.8.3 File Size Monitoring Sensor

132 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*.

133 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*.

8.8.4 File Size Monitoring Sensor Thresholds

- 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.

8.8.5 Device File State Sensor

Device File State Sensor requirements:

- 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.
- The *Device File State Sensor* shall match the monitored *File PDR EntityType*, *EntityInstance*, and *ContainerID* fields to establish its association to the monitored file.
- The *Device File State Sensor* shall implement the DSP0249 PLDM State Set Specification State Set *Device File (68)*.

- 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*.

- For all other *FileClassification* files:
  - 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*

8.8.6 Sensor and File Transfer command interaction

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

Table 5 — Sensor File Transfer Command Interactions

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

8.8.7 The Directory Descriptor Data Model

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.

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

149 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.

150 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*.

8.8.8 File Data Model

In Figure 1 the numeric and state sensors associated with the file match the *EntityType*, *EntityInstance*, and *ContainerID* fields of the *File PDR*.

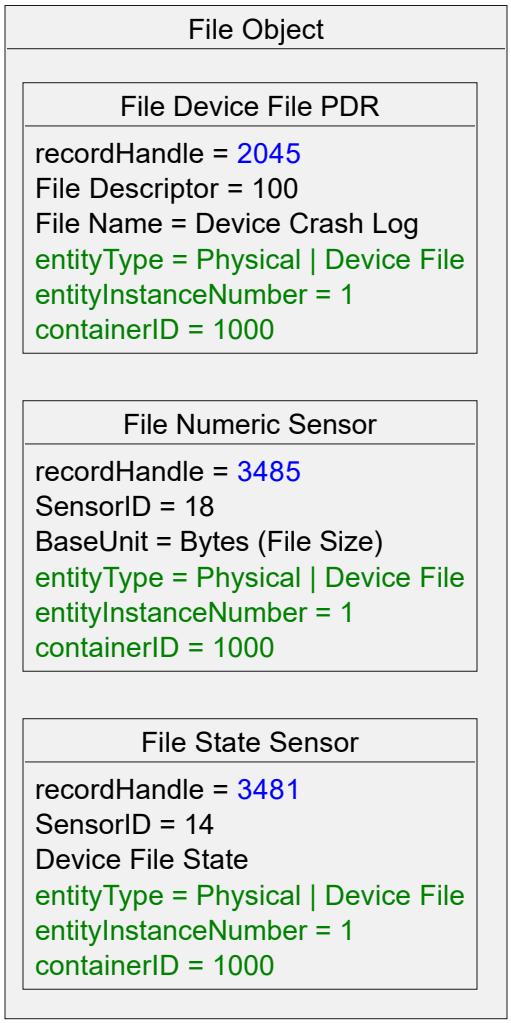
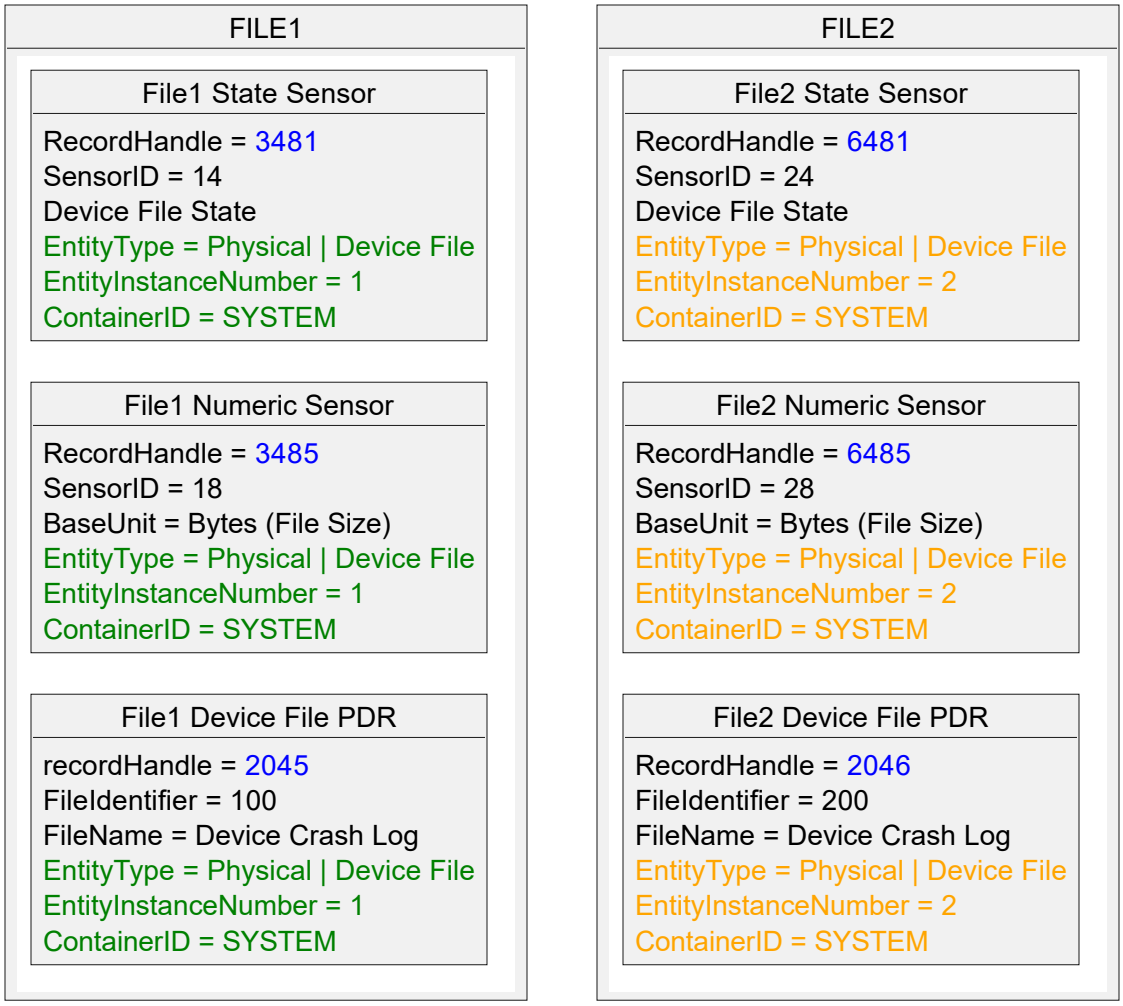


Figure 1 — PLDM for File Transfer File Data Model Implicit Association Example

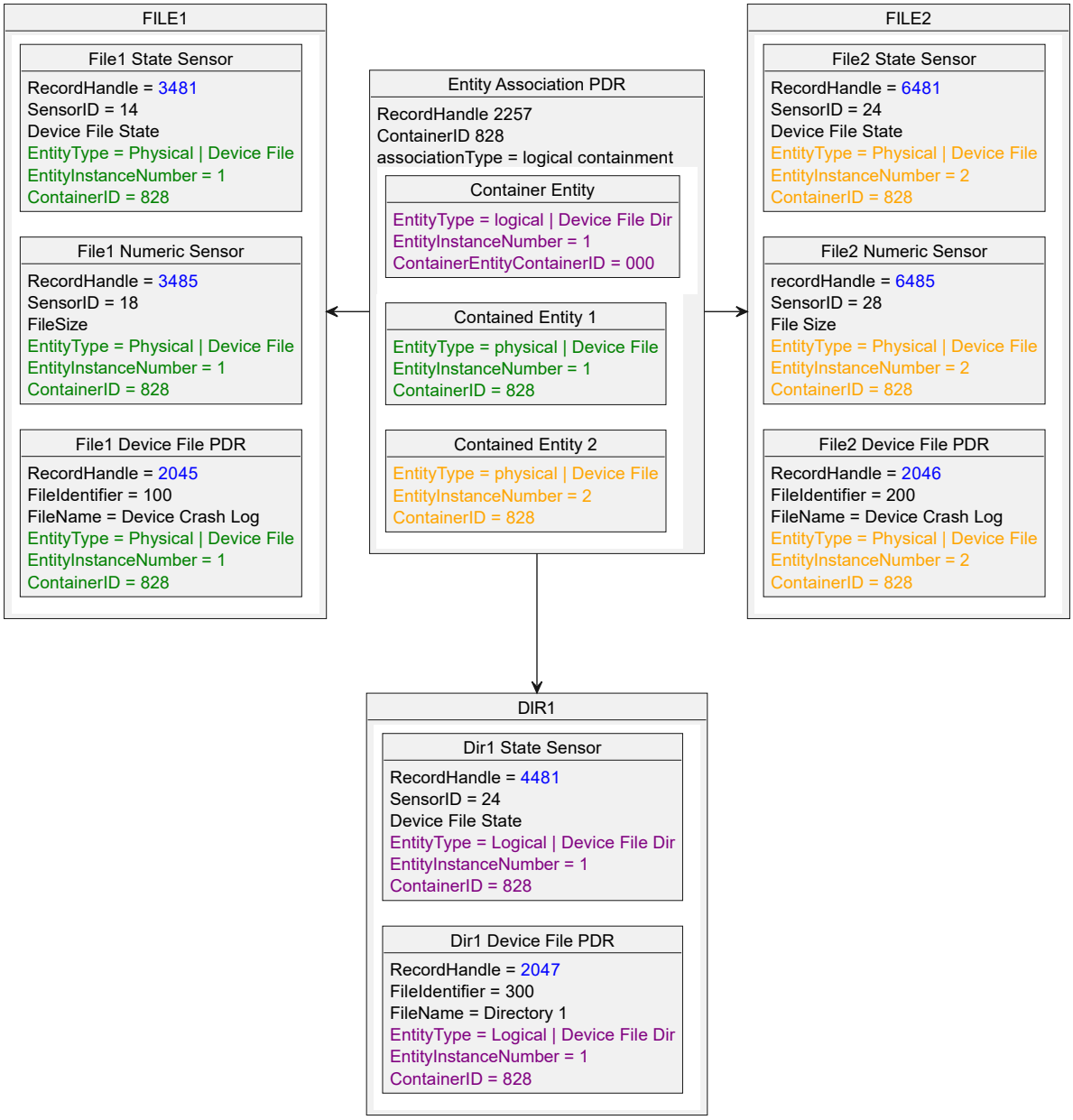
Figure 2 shows a flat file sensor usage example.



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

158 Figure 3 shows a Directory EAR sensor usage example

159



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

9 PLDM for File Transfer Commands

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.

Table 6 — PLDM for File Transfer Command Codes

Command	Code Value	File Host support[3]	File Client support[4]
DfOpen	0x01	Required	Required
DfClose	0x02	Required	Optional
DfHeartbeat	0x03	Optional	Conditional
Reserved	0x04-0x0F		
DfProperties	0x10	Required	Optional
DfGetFileAttribute	0x11	Optional	Optional
DfSetFileAttribute	0x12	Optional	Optional
Reserved	0x13-0x1F		
DfRead MultipartReceive	0x20[1]	Required	Required
DfFIFOSend MultipartSend	0x21[1]	Conditional[2]	Optional
Reserved	0x22-0xFF		

[1] This command value is only to support the DSP0240 GetPLDMCommands command for discovery and is mapped to DSP0240 PLDM Multipart Transfer.

[2] 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.

[3] The File Host is the responder except for the DfFIFOSend command where it is the requester.

[4] The File Client is the requester except for the DfFIFOSend command where it is the responder.

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

166 ## 9.1 DfProperties Command

167 The File Client issues a DfProperties command, as described in Table 7 to list the File Transfer specific capabilities of the File Host.

168 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*.

169 **Table 7 — DfProperties Command Format**

Byte	Type	Request Data
0:3	bitfield32	**DfPropertyAttribute** 1b = Request the specified *DfPropertyAttribute* 0b = *DfPropertyAttribute* not requested [0] — MaxConcurrentMedium [1] — MaxFileDescriptors [2:31] — Reserved

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIBUTE } See Table 16 for values.
1:4	uint32	**DfPropertyAttributeValue** See Table 8.

170 **Table 8 — *DfPropertyAttributeValue* Definition**

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

9.2 DfOpen Command

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.

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.

The returned *FileDescriptor* is used by the File Client for subsequent DfRead, DfHeartbeat, and DfClose commands.

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*.

If the file specified by the *File PDR FileIdentifier* does not exist, the File Host shall return an INVALID_FILE_IDENTIFIER *CompletionCode*.

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*.

9.2.1 DfOpen File Host Pushed requirements

DfOpenPolledPushed set to one (1) (*Pushed*) is only supported if *DfOpenRegFIFO* is set to one (1) (*FIFO*).

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.

9.2.2 DfOpen DfOpenAttribute requirements

File Client / File Host *DfOpenAttribute* requirements:

- 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*.

- 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*.

- 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.

186 - See DfOpen SerialTxFIFO requirements for additional DfOpenAttribute requirements when *File PDR FileClassification* equals DfOpen *SerialTxFIFO*.

187 9.2.3 DfOpen SerialTxFIFO requirements

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

189 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).

190 9.2.4 DfOpen FileDescriptors count requirements

191 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*.

192 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*.

193 9.2.5 File Client file exclusivity usage

194 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*.

195 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*).

196 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".

197 **Table 9 — DfOpen Command Format**

Byte	Type	Request Data
0:1	uint16	**FileIdentifier**

Byte	Type	Request Data
2:3	bitfield16	**DfOpenAttribute** [0] — DfOpenReadWrite, 1b = *Write* (not supported), 0b = *Read* [1] — DfOpenExclusive, 1b = *Exclusive*, 0b = *Non-exclusive* [2] — DfOpenRegFIFO, 1b = *Streaming FIFO* (Serial FIFO), 0b = *Regular* [3] — DfOpenPolledPushed, 1b = *Pushed*, 0b = *Polled* [4:15] Reserved (0)

Byte	Type	Response Data
0	enum8	**CompletionCode** 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 Table 16 for values.
1:2	uint16	**FileDescriptor**

198 ## 9.3 DfClose Command

199 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.

200 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*

201 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*

202 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*.

203 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.

204 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.

205 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.

206 See Implicit File Close under the DfHeartbeat command for information on implicitly closing a file.

207 **Table 10 — DfClose Command Format**

Byte	Type	Request Data
0:1	uint16	**FileDescriptor**

Byte	Type	Request Data
2:3	bitfield16	**DfCloseOptions** 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	**CompletionCode** Possible values: {PLDM_BASE_CODES, INVALID_FILE_DESCRIPTOR, ZEROLENGTH_NOT_ALLOWED, EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED} See Table 16 for values.

9.4 DfGetFileAttribute Command

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

If the *FileIdentifier* requested by the File Client is invalid, the File Host shall return an INVALID_FILE_IDENTIFIER *CompletionCode*.

If the requested *FileIdentifier AttributeReq* is not supported by the File Host, then it shall return an INVALID_DF_ATTRIBUTE *CompletionCode*.

If more than one bit is set in *AttributeReq*, then the File Host shall return an INVALID_DF_ATTRIBUTE *CompletionCode*.

Table 11 — DfGetFileAttribute Command Format

Byte	Type	Request Data
0:1	uint16	**FileIdentifier** This is the *FileIdentifier* returned in the *File PDR* for this file (directory).
2:5	bitfield32	**FileAttributeReq** 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	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIBUTE, INVALID_FILE_IDENTIFIER } See Table 16 for values.
1:4	uint32	**FileAttributeValue** This is a fixed-length return value. See Table 12.

214

Table 12 — DfGetFileAttribute Returned Value Definition

FileAttributeName	Definition
ClientZeroLengthOnly	If the *FileAttributeReq ClientZeroLengthOnly* is set to one (1), then the returned *ClientZeroLengthOnly* 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 *ClientZeroLengthOnly* to zero (0). The file shall be opened exclusively if this attribute is set to one (1).
ChangeIndicator	If the *FileAttributeReq RequestCI* is set to one (1), the returned *ChangeIndicator* is generated by the File Host either at the time of the reception of the *DfGetFileAttribute* 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 *ChangeIndicator* should be a 32-bit CRC.
RequestMaxPoll	If the *FileAttributeReq ReqMaxPoll* is set to one (1), the returned *RequestMaxPoll* is the maximum time, in milliseconds, allowed between reading the File Size Monitoring Sensor or DfRead command before the data may either truncate or wrap, depending on the *File PDR FileCapabilities* settings.

9.5 DfSetFileAttribute Command

The DfSetFileAttribute command, as described in Table 13, is used by the File Client to set specific dynamic file attributes such as file preservation.

If the *FileIdentifier* requested by the File Client is invalid, the File Host shall return an INVALID_FILE_IDENTIFIER *CompletionCode*.

If the requested *FileIdentifier AttributeReq* is not supported by the File Host, then it shall return an INVALID_DF_ATTRIBUTE *CompletionCode*.

If more than one bit is set in *AttributeReq*, then the File Host shall return an INVALID_DF_ATTRIBUTE *CompletionCode*.

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*.

If the SUCCESS *CompletionCode* is not returned, then no change is made to the DfSetFileAttribute.

222

Table 13 — DfSetFileAttribute Command Format

Byte	Type	Request Data
0:1	uint16	**FileIdentifier** This is the *FileIdentifier* returned in the *File PDR* for this file (directory).
2:3	bitfield16	**FileAttributeSet** 1b = Request setting specified attribute 0b = setting of attribute not requested [0] ClientZeroLengthOnly [1:15] Reserved (0)
4:7	uint32	**FileAttributeValue** This is a fixed-length value. See Table 14.

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIBUTE, INVALID_FILE_IDENTIFIER, FILE_OPEN } See Table 16 for values.

223

Table 14 — DfSetFileAttribute SUCCESS Value Definition

FileAttributeName	Definition
ClientZeroLengthOnly	If *ClientZeroLengthOnly FileAttributeValue* 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 *ClientZeroLengthOnly FileAttributeValue* to zero (0) allows the file to be deleted or updated by the File Host. The file shall be subsequently opened exclusively if this attribute is set to one. If the *ClientZeroLengthOnly FileAttributeValue* bit is set to (1), the File Host shall not update or add to the file after initial creation. The File Client is only allowed to change the *ClientZeroLengthOnly FileAttributeValue* if the file is not currently open. If the file is currently open, the File Host and the File Client shall not change the ClientZeroLengthOnly state and the File Host shall return the FILE_OPEN *CompletionCode*. The *ClientZeroLengthOnly* state shall be preserved across resets and power cycles for *non-volatile* files *File PDR FileCapabilities DataVolatility*. The *ClientZeroLengthOnly* state for *volatile* files should be preserved across resets.

9.6 DfHeartbeat Command

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

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.

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.

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.

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*.

9.6.1 Implicit File Close

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.

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.

233 **Table 15 — DfHeartbeat Command Format**

Byte	Type	Request Data
0:1	uint16	**FileDescriptor**
2:5	uint32	**RequesterMaxInterval** The requested maximum supported *NegotiatedInterval* in milliseconds

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_FILE_DESCRIPTOR } See Table 16 for values.
1:4	uint32	**ResponderMaxInterval** The maximum supported *NegotiatedInterval* in milliseconds from the responder

234 ## 9.7 Error Completion Codes

235 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.

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

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

9.8 DfRead (DSP0240 MultipartReceive)

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*.

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.

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*.

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.

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.

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*.

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

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

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

9.8.1 Serial FIFO type file characteristics

Files classified as a *SerialTxFIFO* have specific characteristics, similar to an endpoint streaming data of a Universal Asynchronous Receiver-Transmitter (UART) device. See DSP0240 for the terms *Transfer Part*, *Section*, *Transferred Part Size*, and *Negotiated Transfer Part Size*. The following requirements apply: with the mappings defined in Table 19.

- Seeking is not supported (MultipartReceive *Requested SectionOffset* shall be set to zero).
- Single *Transfer 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 should issue the MultipartReceive to read a *SerialTxFIFO* type file until the current *Transferred Part Size* is less than the *Negotiated Transfer Part Size*, indicating that all the available data has been transferred. There are other methods, such as polling the File Size Sensor in between each MultipartReceive command, but using the *Transferred Part Size* method is the most efficient.
- See DfOpen *SerialTxFIFO* requirements for additional DfOpenAttribute requirements.

249 The DfRead command is implemented using the *Multipart Transfer Commands*, as Table 6 describes.

250 9.8.2 DfRead command details

251 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.

252 DfRead File Client request requirements are detailed in Table 17:

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

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

254 DfRead File Host response requirements are detailed in Table 18:

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

MultipartReceive TransferOperation	All File Types	Additional *SerialTxFIFO* File Type Requirements
XFER_FIRST_PART		The File Host shall respond with a *TransferFlag* equal to START_AND_END, as required by DSP0240 when *Transferred Part Size* is less than or equal to the *Negotiated Transfer Part Size*.
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.

256 **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	**FileDescriptor**	TransferContext
6:9	uint32	Initially 0	DataTransferHandle
10:13	uint32	**FileOffset**	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

257 9.9 DfFIFOSend (DSP0240 MultipartSend)

258 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)

259 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.

260 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.

261 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.

262 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.

263 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.

264 The File Host shall make:

- *SectionLengthBytes* equal to *DataLengthBytes*, and
- *Transferred Part Size* equal to or less than the *Negotiated Transfer Part Size* from the most recent successfully completed NegotiateTransferParameters command.

265 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.

266 **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	**FileDescriptor**	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)

267

10 ANNEX A (informative) Change Log

Version	Date	Description
1.0.0	2024-07-29	Initial Release
1.0.1	2026-01-07	- Added IETF and ISO registered trademarks - File and Directory Discovery - Added, "if it is the event receiver" to when the File Client may enable PLDM events - Table 4 - File Sensors and Thresholds - SerialTxFIFO - aligned text with DSP0249 - Table 21 - SerialTxFIFO - Removed Device File State Sensor entries to match text - Serial FIFO type file characteristics - Align the terms to those used in DSP0240 - Corrected the formula the File Client should use to transfer all the available data - Annex D - Added a MultipartReceive and MultipartSend maximum part byte count calculation example - Reformatted ClientZeroLengthOnly Usage NOTE to not generate paragraph numbers - Changed Directory EAR diagram to use logical to match text

268

11 ANNEX B (informative) Sensor Threshold Event Examples

269 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*.

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

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

FileClassification	FMS	PFS CFS	WH	CH	FH	DFSS	FSMS Event	Note
SerialTxFIFO	256	0 1	128	230	256			
SerialTxFIFO	256	1 1	128	230	256			
SerialTxFIFO	256	1 1	128	230	256			
SerialTxFIFO	256	1 200	128	230	256		WH	
SerialTxFIFO	256	256 256	128	230	256	FDhW	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 *Static File*
FRUDataFile OtherFile	1024	1024 1024				FiU		existing *Fixed Length File* with update
BootLog DiagnosticLog SecurityLog OtherLog	1024	100 101	512	920	1024	FiU		Crossed WH

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

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

273 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*

274 FSMS Event = *File Size Monitoring Sensor* Event

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

12 ANNEX C (informative) File PDR FileClassification FileCapabilities Examples

277 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.

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

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

FileClassification	*FileCapabilities*	DfOpen Attributes	Description
BootLog DiagnosticLog SecurityLog FRUDataFile OtherLog OtherFile	ExReadOpen=0 FileTrunc=0 (*Truncate*) DataType=0 (*Regular*) Polled=1 (*Polled*) Pushed=0 DataVolatility=1 (*Non-volatile*) FileModify=0 (*Append*) FcZeroLengthPermitted=0 FcWritesPermitted=0	DfOpenReadWrite=*Read* DfOpenExclusive=*Non-exclusive* DfOpenRegFIFO=*Regular* DfOpenPolledPushed=*Polled*	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 *ZeroLength*=1 option.

13 ANNEX D (informative) PLDM for File Transfer Examples

This informative section describes typical File Transfer command usage.

13.1 Example calculation of maximum number of *Data* bytes per MultipartReceive *Transfer Part*

The maximum number of *Data* bytes that can be transferred in a single MultipartReceive *Transfer Part* can be calculated by starting with the *Negotiated Transfer Size* and subtracting the sum of the bytes in that *Transfer Part* that are not *Data*.

Table 23 shows how to calculate the maximum number of *Data* bytes per MultipartReceive *Transfer Part* for an example *Negotiated Transfer Size* of 256 bytes.

Table 23 — Maximum number of *Data* bytes per MultipartReceive *Transfer Part* calculation example

Negotiated Transfer Size	–	Bytes that are not *Data*	=	Maximum number of *Data* bytes
256 bytes	–	Sum of: • *(3 bytes) PLDM Message Header* • *(1 byte) CompletionCode* • *(1 byte) TransferFlag* • *(4 bytes) DataTransferHandle* • *(4 bytes) DataLengthBytes* • *(4 bytes) DataIntegrityChecksum* 17 bytes	=	239 bytes

13.2 Example calculation of maximum number of *Data* bytes per MultipartSend *Transfer Part*

The maximum number of *Data* bytes that can be transferred in a single MultipartSend *Transfer Part* can be calculated by starting with the *Negotiated Transfer Size* and subtracting the sum of the bytes in that *Transfer Part* that are not *Data*.

Table 24 shows how to calculate the maximum number of *Data* bytes per MultipartSend *Transfer Part* for an example *Negotiated Transfer Size* of 256 bytes.

288 **Table 24 — Maximum number of *Data* bytes per MultipartSend *Transfer Part* calculation example**

Negotiated Transfer Size	**−**	**Bytes that are not *Data***	**=**	**Maximum number of *Data* bytes**
256 bytes	−	Sum of: • *(3 bytes) PLDM Message Header* • *(1 byte) PLDMType* • *(1 byte) TransferFlag* • *(4 bytes) TransferContext* • *(4 bytes) DataTransferHandle* • *(4 bytes) SectionOffset* • *(4 bytes) SectionLengthBytes* • *(4 bytes) DataLengthBytes* • *(4 bytes) DataIntegrityChecksum* 29 bytes	=	227 bytes

289 # 13.3 PLDM for File Transfer flow examples

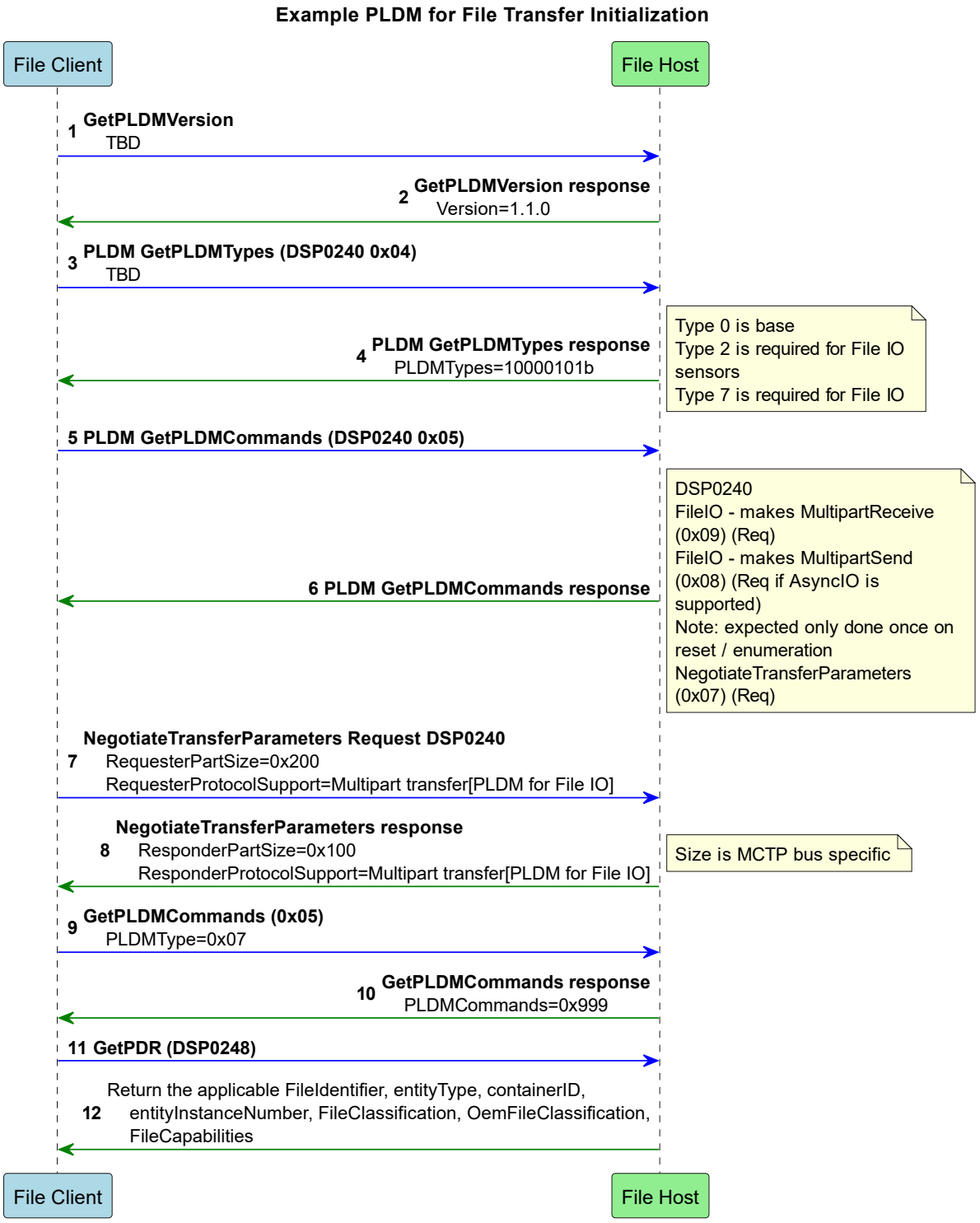
290 This section gives examples of typical flows involving various File Transfer commands.

- Initialization example
- Regular log file read example
- Polled Serial log read example
- Pushed Serial log read example

291 ## 13.3.1 PLDM for File Transfer initialization example

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

293



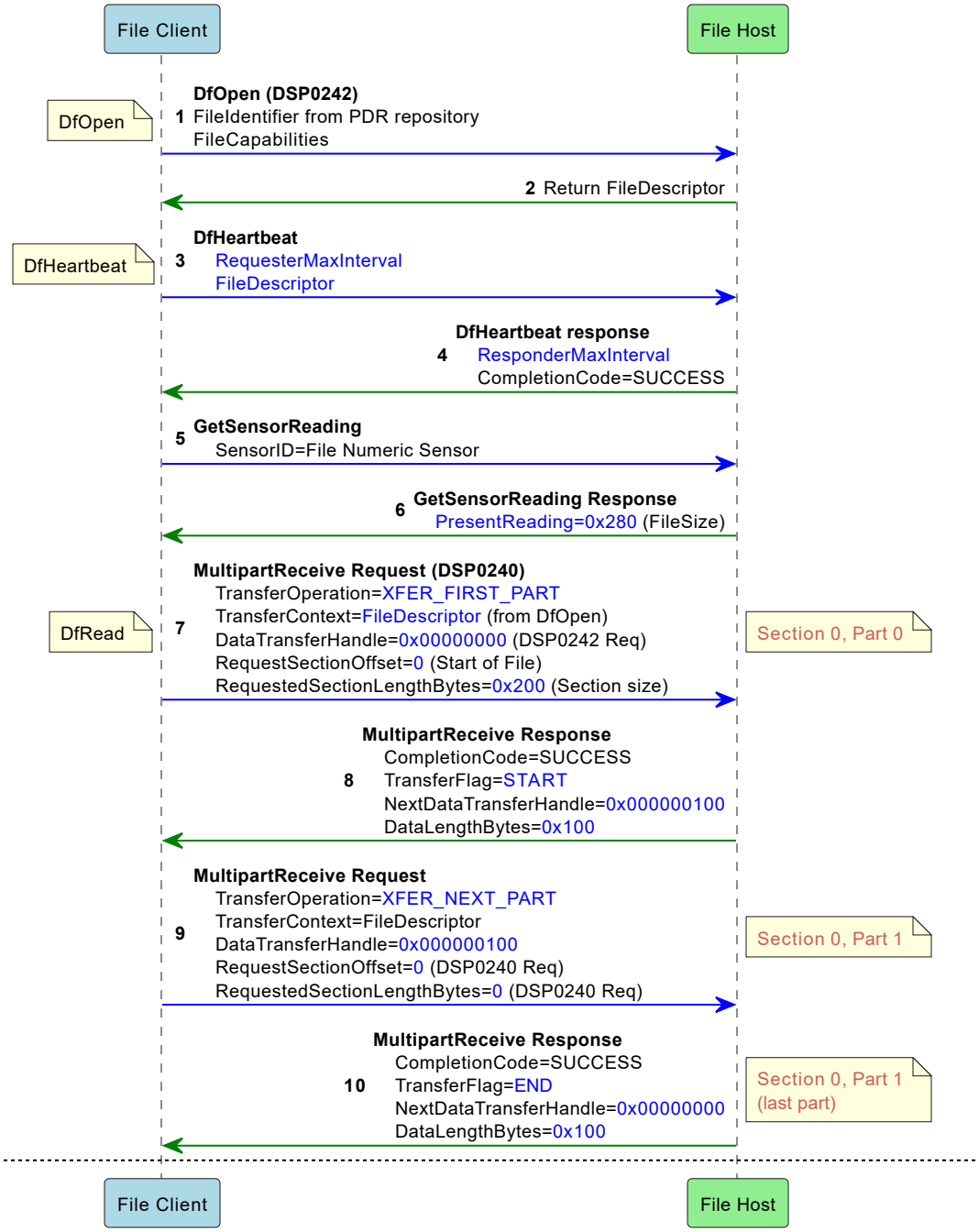
294 **Figure 4 — PLDM for File Transfer Initialization Example**

295 ### 13.3.2 Regular log file read example

296 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.

297



298 **Figure 5 — *Regular* Log File Read Example - Page 1**

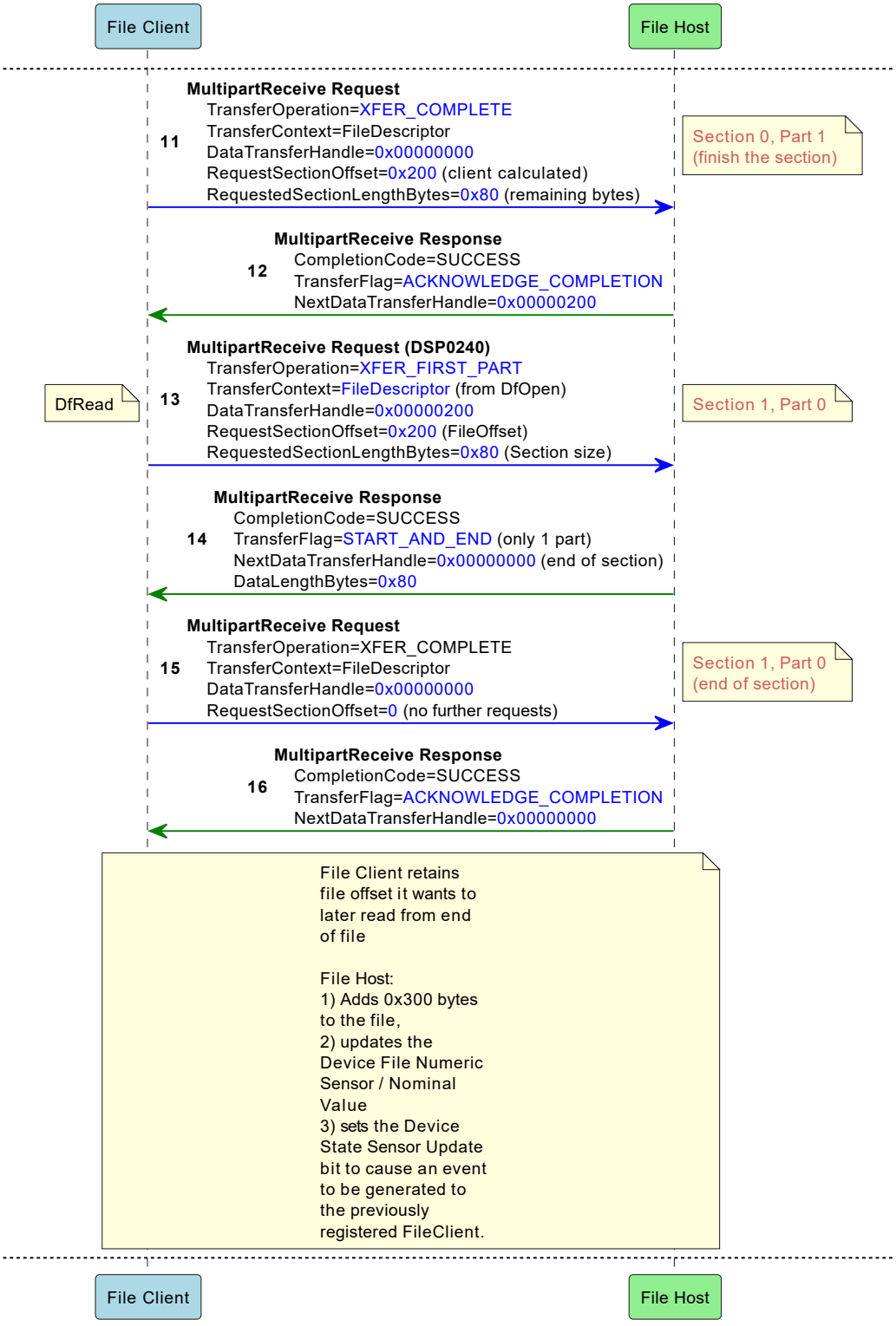


Figure 6 — *Regular* Log File Read Example - Page 2

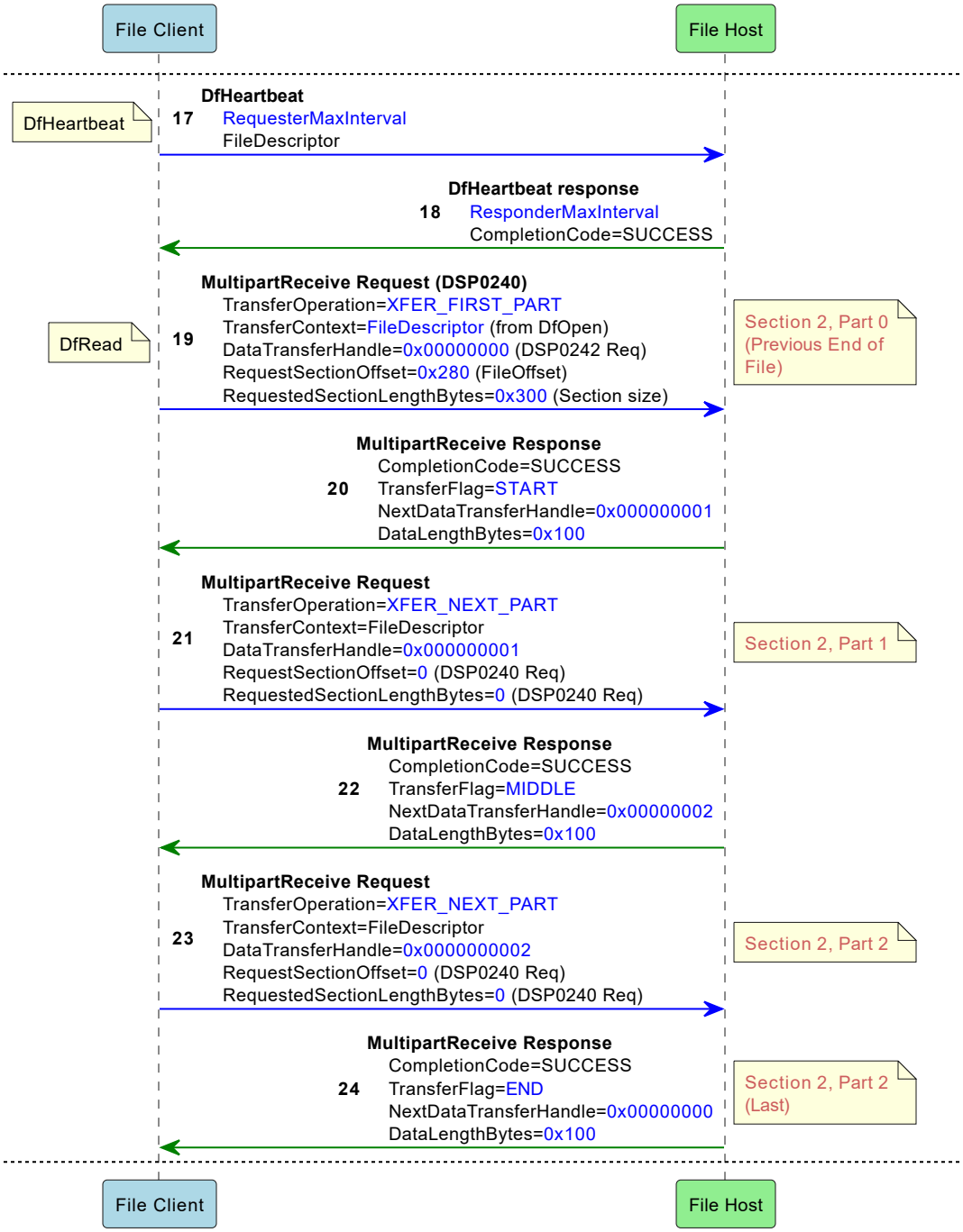
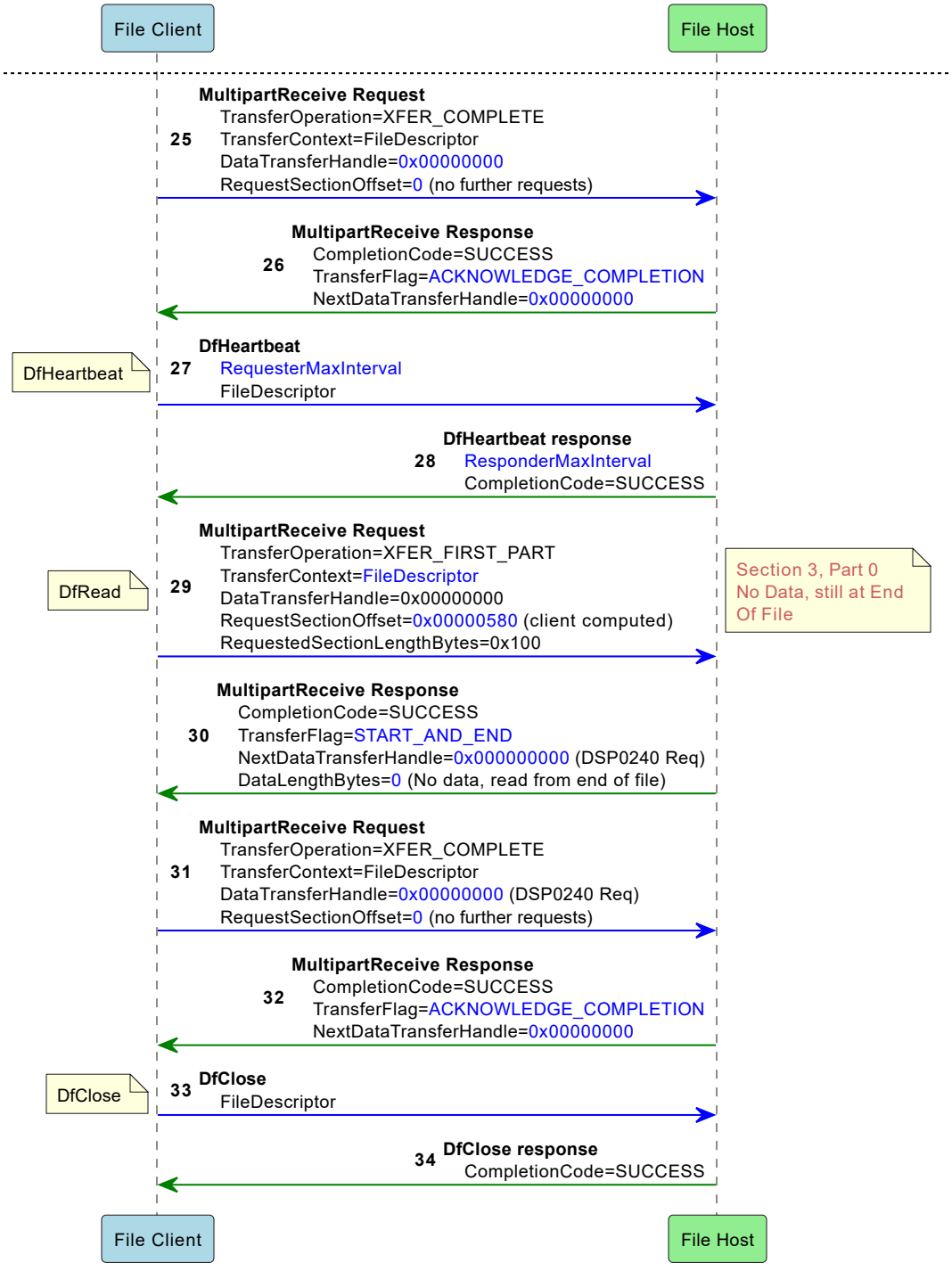


Figure 7 — *Regular* Log File Read Example - Page 3

303



304

Figure 8 — *Regular* Log File Read Example - Page 4

13.3.3 Polled Serial Log read example

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

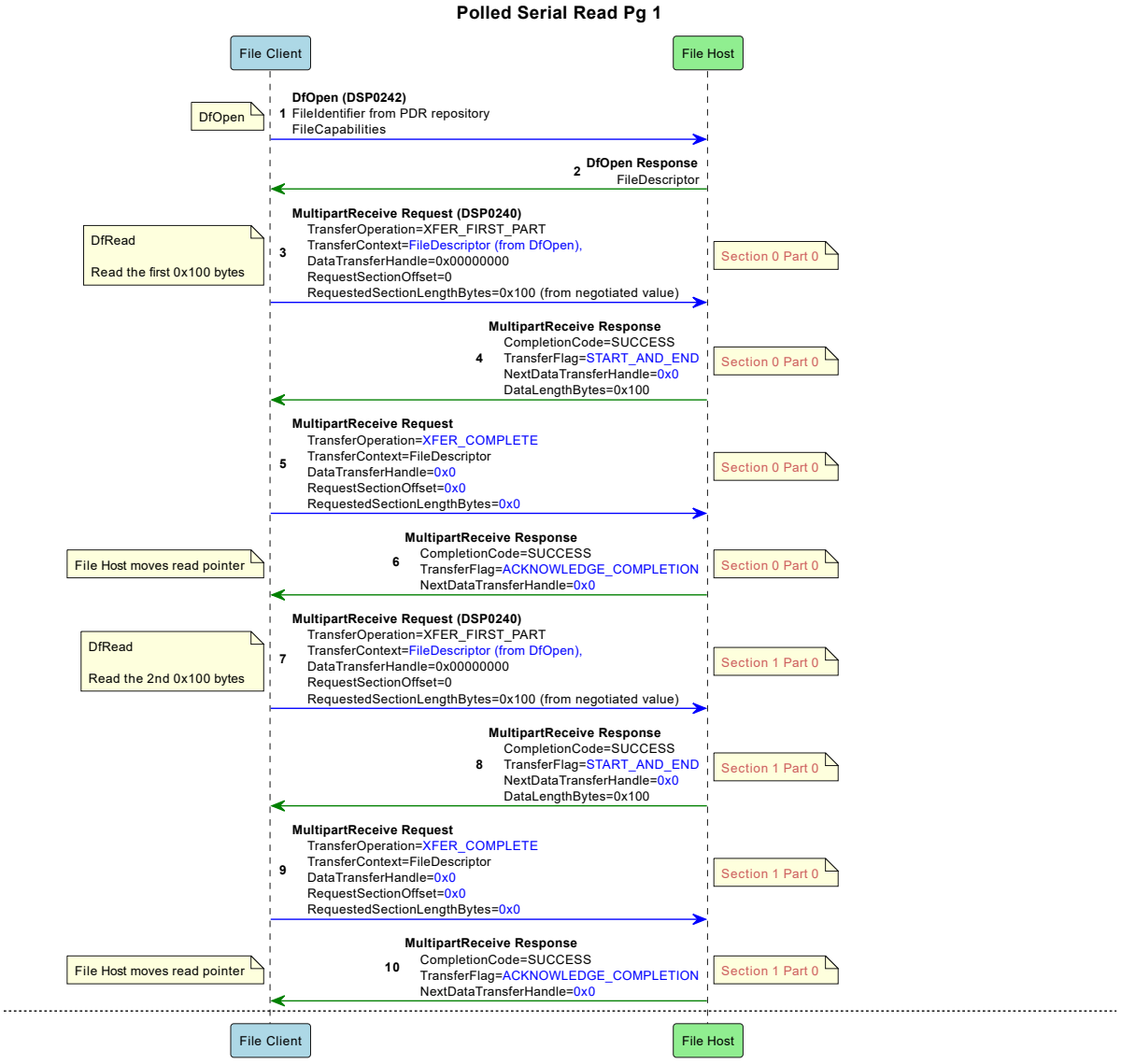
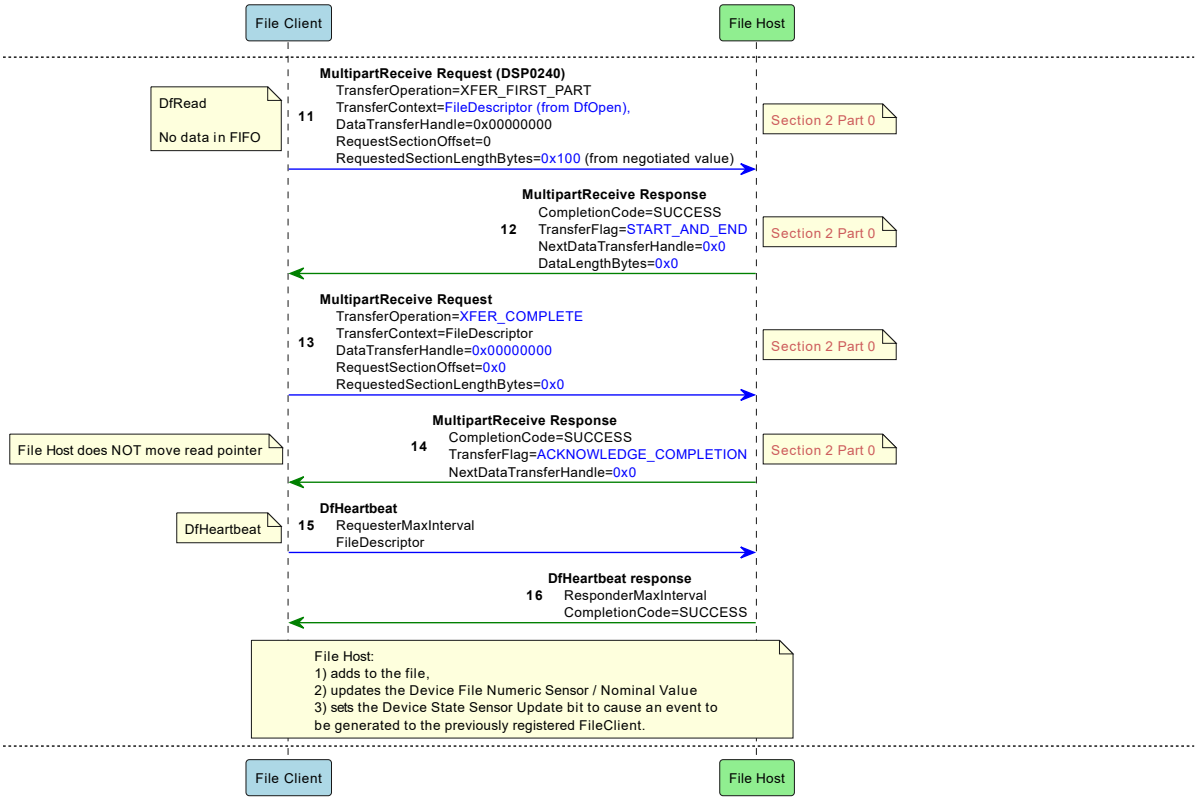


Figure 9 — Polled Serial Read Example - Page 1

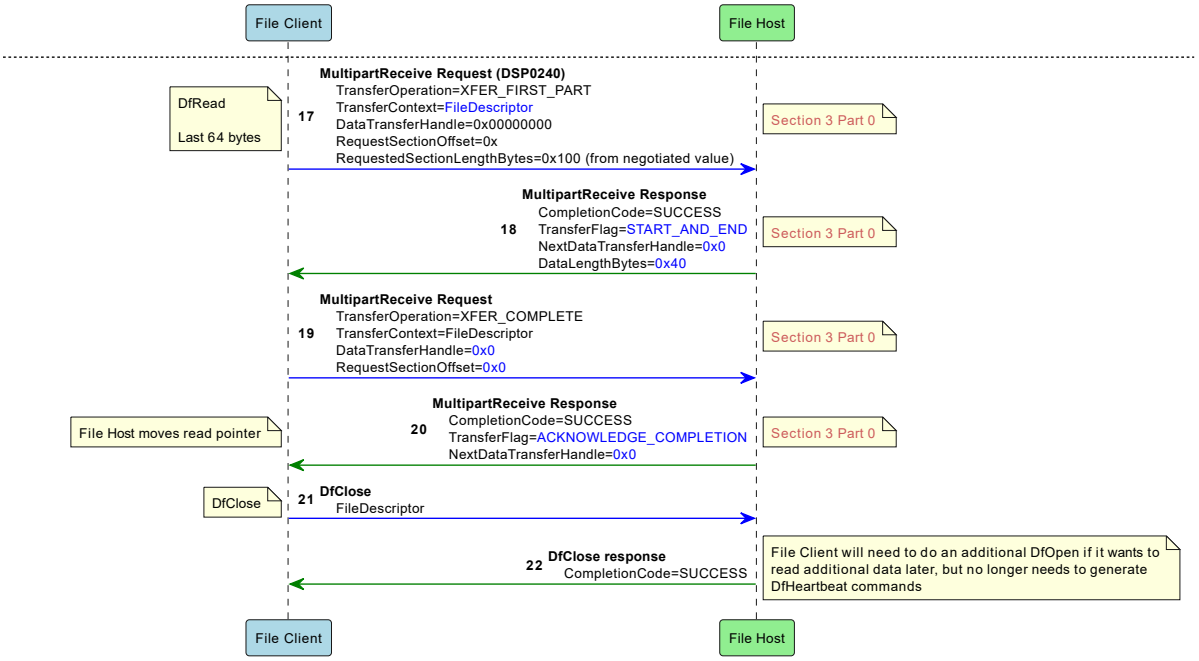
309



310

Figure 10 — Polled Serial Read Example - Page 2

311

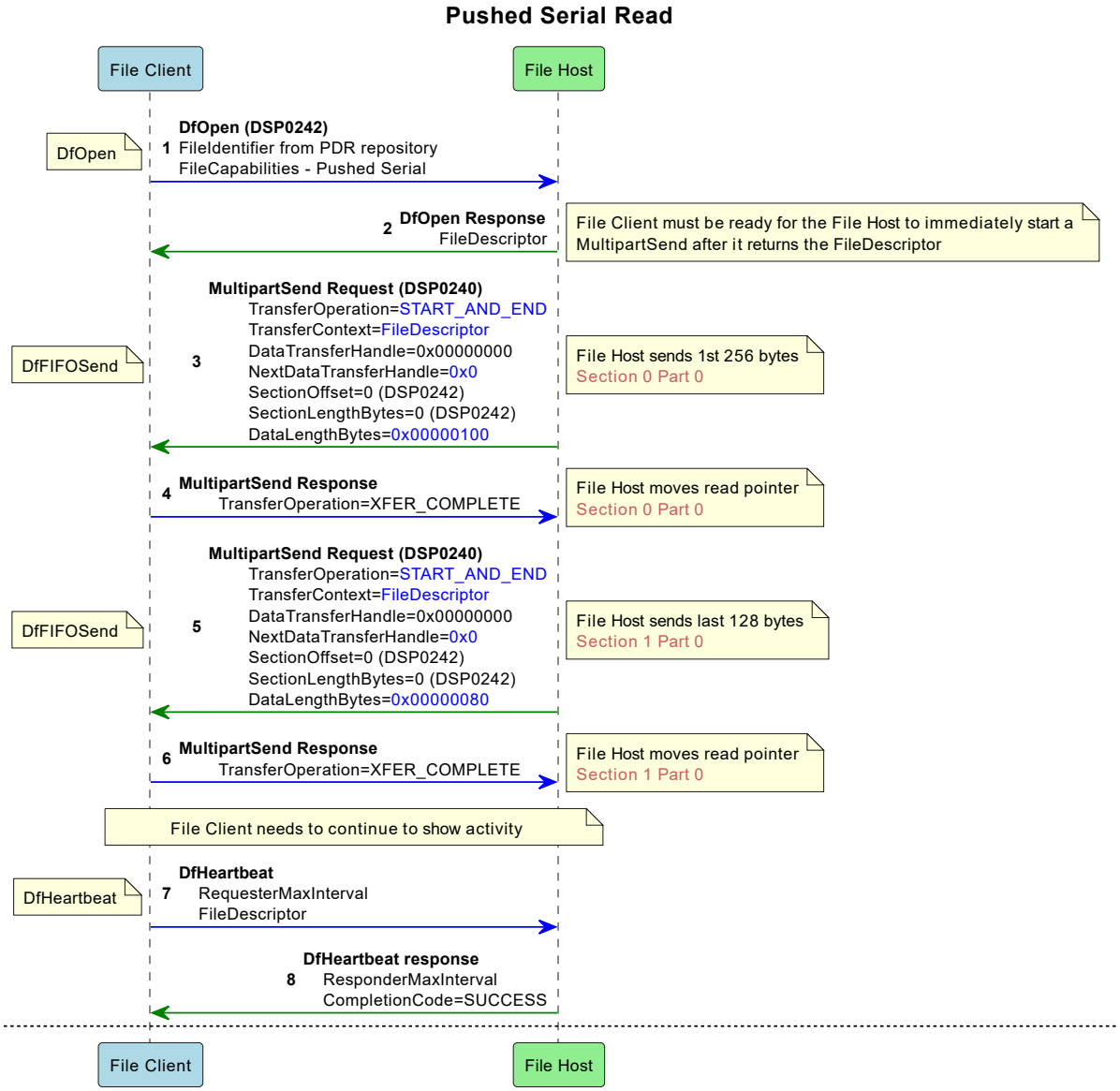


312 **Figure 11 — Polled Serial Read Example - Page 3**

313 ### 13.3.4 Pushed Serial Log read example

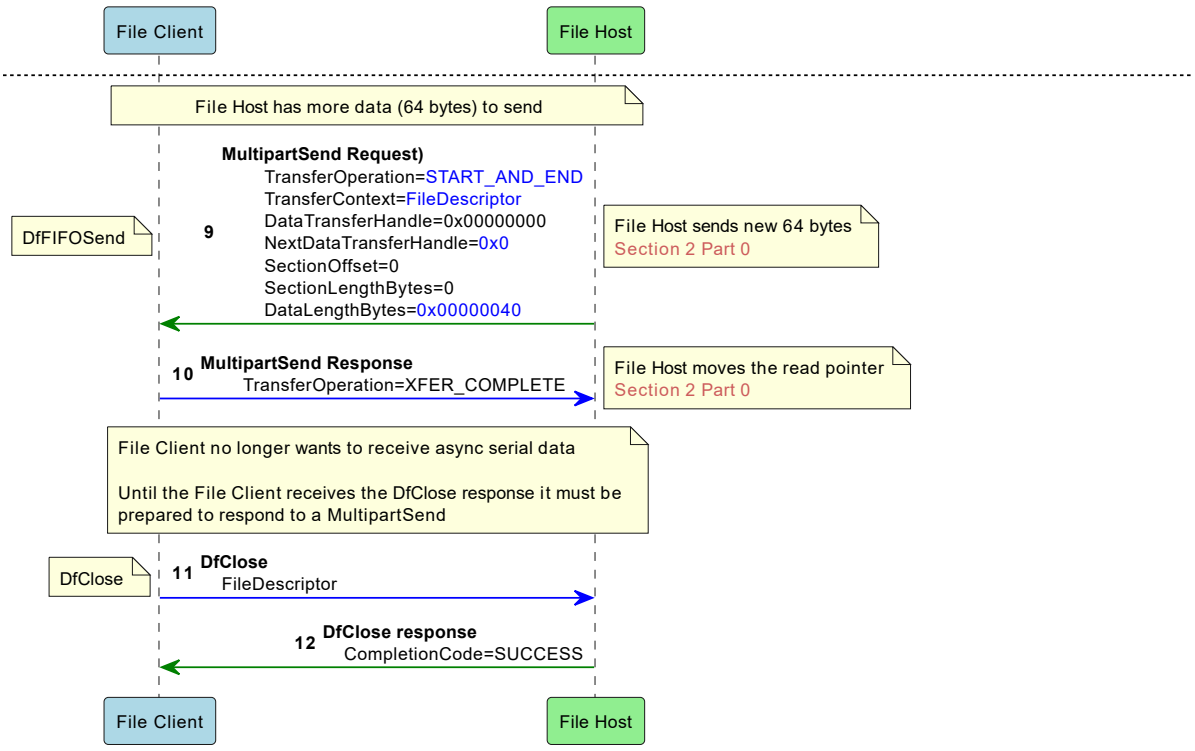
314 Figures 12 and 13 show an example of a *Pushed SerialTxFIFO* log read.

315



316 **Figure 12 — Pushed SerialTxFIFO Log Read Example - Page 1**

317



318 **Figure 13 — Pushed SerialTxFIFO Log Read Example - Page 2**