Platform Level Data Model (PLDM) for File Transfer Specification

Supersedes: None

Document Class: Normative

Document Status: Published

Document Language: en-US
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.

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patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a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

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.

This document’s normative language is English. Translation into other languages is permitted.
CONTENTS

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 DIProperties 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
1 Foreword

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

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

1.1 Acknowledgments

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.
3 Scope

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

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

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.

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

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://dmtf.org/sites/default/files/standards/documents/DSP0248_1.3.pdf

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

- **DMTF DSP1001, Management Profile Specification Usage Guide 1.1**

- **DMTF DSP4014, DMTF Process for Working Bodies 2.13**

- **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
7 PLDM for File Transfer version

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).
8 PLDM for File Transfer Concepts

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.

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.

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.

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.

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.

8.1 File Metadata

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.

- **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.

- **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.

### 8.3.2 File Types and Classification

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.

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.

#### Table 1 — FileClassification Examples

<table>
<thead>
<tr>
<th>FileClassification</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BootLog</td>
<td>Typically holds device initialization data (events) but has no additional entries after initialization completes</td>
</tr>
<tr>
<td>SerialTxFIFO</td>
<td>Typically removes the data after successful transfer to the receiver or if the FIFO overflows</td>
</tr>
<tr>
<td>DiagnosticLog</td>
<td>Typically a variable length file where data can be appended until maximum storage limit is reached</td>
</tr>
</tbody>
</table>
87

### FileClassification

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

88

#### 8.3.3 File and Directory Discovery

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.

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.

90

#### 8.3.4 File System Hierarchy Discovery

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.

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

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

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:
   - Updates *File Size* with the current file size
   - Updates the **Device File State Sensor** to *File is Updated*
   - 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:
   - Register for events from the **File Size Monitoring Sensor**
   - Register for events from the **Device File State Sensor**
   - Poll the **File Size Monitoring Sensor** looking for a nonzero *File Size* value
   - 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.

---

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

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.
8.8.1 The File Descriptor Data Model

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

<table>
<thead>
<tr>
<th>File Characteristic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static File</td>
<td>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</td>
</tr>
<tr>
<td>Fixed Length File</td>
<td>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</td>
</tr>
<tr>
<td>Regular File</td>
<td>READ, MODIFY, or APPEND after creation, including the optional DfClose ZeroLength option</td>
</tr>
<tr>
<td>SerialTxFIFO</td>
<td>A FIFO file whose file size may grow or shrink</td>
</tr>
</tbody>
</table>

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

Table 3 — Sensor Usage for Different File Characteristics

<table>
<thead>
<tr>
<th>File Characteristic</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static File</td>
<td>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.</td>
</tr>
<tr>
<td>Fixed Length File</td>
<td>The Device File State Sensor is used to indicate and/or notify of a file update.</td>
</tr>
<tr>
<td>Regular File</td>
<td>The FatalHigh threshold is used to allow overflow detection. Device File State Sensor is used to indicate and/or notify of a file update.</td>
</tr>
<tr>
<td>File Characteristic</td>
<td>Usage</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| 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.

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Static File</td>
<td>Should</td>
<td>Should not</td>
<td>Should</td>
<td>Should not support <strong>File is at Maximum File State</strong></td>
</tr>
<tr>
<td>Fixed Length File</td>
<td>Should</td>
<td>Should not</td>
<td>Shall</td>
<td>Should not support <strong>File is at Maximum File State</strong></td>
</tr>
<tr>
<td>Regular File</td>
<td>Shall</td>
<td>Should</td>
<td>Shall</td>
<td>Should support all defined states</td>
</tr>
<tr>
<td>SerialTxFIFO</td>
<td>Shall</td>
<td>Shall</td>
<td>Shall</td>
<td>Should not support <strong>File is Updated or File is not updated</strong></td>
</tr>
</tbody>
</table>

8.8.3 File Size Monitoring Sensor

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.

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

### 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.

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

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 Entity Type* and *Entity Instance* 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.

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`.

<table>
<thead>
<tr>
<th>File Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Device File PDR</strong></td>
</tr>
<tr>
<td>recordHandle = 2045</td>
</tr>
<tr>
<td>File Descriptor = 100</td>
</tr>
<tr>
<td>File Name = Device Crash Log</td>
</tr>
<tr>
<td>entityType = Physical</td>
</tr>
<tr>
<td>entityInstanceNumber = 1</td>
</tr>
<tr>
<td>containerID = 1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Numeric Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordHandle = 3485</td>
</tr>
<tr>
<td>SensorID = 18</td>
</tr>
<tr>
<td>BaseUnit = Bytes (File Size)</td>
</tr>
<tr>
<td>entityType = Physical</td>
</tr>
<tr>
<td>entityInstanceNumber = 1</td>
</tr>
<tr>
<td>containerID = 1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File State Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordHandle = 3481</td>
</tr>
<tr>
<td>SensorID = 14</td>
</tr>
<tr>
<td>Device File State</td>
</tr>
<tr>
<td>entityType = Physical</td>
</tr>
<tr>
<td>entityInstanceNumber = 1</td>
</tr>
<tr>
<td>containerID = 1000</td>
</tr>
</tbody>
</table>

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

Figure 2 shows a flat file sensor usage example.
### FILE1

**File1 State Sensor**
- RecordHandle = 3481
- SensorID = 14
- Device File State
- EntityType = Physical | Device File
- EntityInstanceNumber = 1
- ContainerID = SYSTEM

**File1 Numeric Sensor**
- RecordHandle = 3485
- SensorID = 18
- BaseUnit = Bytes (File Size)
- EntityType = Physical | Device File
- EntityInstanceNumber = 1
- ContainerID = SYSTEM

**File1 Device File PDR**
- RecordHandle = 2045
- FileIdentifier = 100
- FileName = Device Crash Log
- EntityType = Physical | Device File
- EntityInstanceNumber = 1
- ContainerID = SYSTEM

### FILE2

**File2 State Sensor**
- RecordHandle = 6481
- SensorID = 24
- Device File State
- EntityType = Physical | Device File
- EntityInstanceNumber = 2
- ContainerID = SYSTEM

**File2 Numeric Sensor**
- RecordHandle = 6485
- SensorID = 28
- BaseUnit = Bytes (File Size)
- EntityType = Physical | Device File
- EntityInstanceNumber = 2
- ContainerID = SYSTEM

**File2 Device File PDR**
- RecordHandle = 2046
- FileIdentifier = 200
- FileName = Device Crash Log
- EntityType = Physical | Device File
- EntityInstanceNumber = 2
- ContainerID = SYSTEM

---

**Figure 2 — PLDM for File Transfer Flat File Sensor Usage Example**
Figure 3 shows a Directory EAR sensor usage example.

**FILE1**

File1 State Sensor
- RecordHandle = 3481
- SensorID = 14
- Device File State
  - Entity Type = Physical | Device File
  - Entity Instance Number = 1
  - Container ID = 828

File1 Numeric Sensor
- RecordHandle = 3485
  - SensorID = 18
  - File Size
    - Entity Type = Physical | Device File
    - Entity Instance Number = 1
    - Container ID = 828

File1 Device File PDR
- RecordHandle = 2045
  - File Identifier = 100
  - Filename = Device Crash Log
    - Entity Type = Physical | Device File
    - Entity Instance Number = 1
    - Container ID = 828

**FILE2**

File2 State Sensor
- RecordHandle = 6481
  - SensorID = 24
  - Device File State
    - Entity Type = Physical | Device File
    - Entity Instance Number = 2
    - Container ID = 828

File2 Numeric Sensor
- RecordHandle = 6485
  - SensorID = 28
  - File Size
    - Entity Type = Physical | Device File
    - Entity Instance Number = 2
    - Container ID = 828

File2 Device File PDR
- RecordHandle = 2046
  - File Identifier = 200
  - Filename = Device Crash Log
    - Entity Type = Physical | Device File
    - Entity Instance Number = 2
    - Container ID = 828

**DIR1**

Dir1 State Sensor
- RecordHandle = 4481
  - SensorID = 24
  - Device File State
    - Entity Type = Physical | Device File Dir
    - Entity Instance Number = 1
    - Container ID = 828

Dir1 Device File PDR
- RecordHandle = 2047
  - File Identifier = 300
  - Filename = Directory 1
    - Entity Type = Physical | Device File Dir
    - Entity Instance Number = 1
    - Container ID = 828

**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

<table>
<thead>
<tr>
<th>Command</th>
<th>Code Value</th>
<th>File Host support</th>
<th>File Client support</th>
</tr>
</thead>
<tbody>
<tr>
<td>DfOpen</td>
<td>0x01</td>
<td>Required</td>
<td>Required</td>
</tr>
<tr>
<td>DfClose</td>
<td>0x02</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>DfHeartbeat</td>
<td>0x03</td>
<td>Optional</td>
<td>Conditional</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x04-0x0F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DfProperties</td>
<td>0x10</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>DfGetFileAttribute</td>
<td>0x11</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>DfSetFileAttribute</td>
<td>0x12</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x13-0x1F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DfRead MultipartReceive</td>
<td>0x20¹</td>
<td>Required</td>
<td>Required</td>
</tr>
<tr>
<td>DfFIFOSend MultipartSend</td>
<td>0x21¹</td>
<td>Conditional²</td>
<td>Optional</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x22-0xFF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
9.1 DfProperties Command

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

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.

Table 7 — DfProperties Command Format

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Request Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:3</td>
<td>bitfield32</td>
<td>DfPropertyAttribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1b = Request the specified DfPropertyAttribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0b = DfPropertyAttribute not requested</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[0] — MaxConcurrentMedium</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1] — MaxFileDescriptors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[2:31] — Reserved</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Response Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>enum8</td>
<td>CompletionCode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIBUTE }</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Table 16 for values.</td>
</tr>
<tr>
<td>1:4</td>
<td>uint32</td>
<td>DfPropertyAttributeValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Table 8.</td>
</tr>
</tbody>
</table>

Table 8 — DfPropertyAttributeValue Definition

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

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

### 9.2.3 DfOpen SerialTxFIFO requirements

Requirements when File PDR FileClassification equals DfOpen SerialTxFIFO:

- 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).

### 9.2.4 DfOpen FileDescriptors count requirements

- 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.
- 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.

### 9.2.5 File Client file exclusivity usage

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.

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).

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”.

### Table 9 — DfOpen Command Format

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Request Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:1</td>
<td>uint16</td>
<td>FileIdentifier</td>
</tr>
<tr>
<td>Byte</td>
<td>Type</td>
<td>Request Data</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>2:3</td>
<td>bitfield16</td>
<td>DfOpenAttribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[0] — DfOpenReadWrite, 1b = Write (not supported), 0b = Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1] — DfOpenExclusive, 1b = Exclusive, 0b = Non-exclusive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[2] — DfOpenRegFIFO, 1b = Streaming FIFO (Serial FIFO), 0b = Regular</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[3] — DfOpenPolledPushed, 1b = Pushed, 0b = Polled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[4:15] Reserved (0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Response Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>enum8</td>
<td>CompletionCode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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 }</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Table 16 for values.</td>
</tr>
<tr>
<td>1:2</td>
<td>uint16</td>
<td>FileDescriptor</td>
</tr>
</tbody>
</table>
9.3 DfClose Command

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.

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

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

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.

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.

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.

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.

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

Table 10 — DfClose Command Format

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Request Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:1</td>
<td>uint16</td>
<td>FileDescriptor</td>
</tr>
</tbody>
</table>
### Request Data

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>DfCloseOptions</th>
</tr>
</thead>
</table>
| 2:3  | bitfield16 | 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) |

### Response Data

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>CompletionCode</th>
</tr>
</thead>
</table>
| 0    | enum8 | 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

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Request Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:1</td>
<td>uint16</td>
<td>FileIdentifier&lt;br&gt;This is the FileIdentifier returned in the File PDR for this file (directory).</td>
</tr>
<tr>
<td>2:5</td>
<td>bitfield32</td>
<td>FileAttributeReq&lt;br&gt;1b = Request the specified current attribute status&lt;br&gt;0b = Attribute status not requested&lt;br&gt;[0] — ClientZeroLengthOnly&lt;br&gt;[1:15] Reserved (0)&lt;br&gt;[16] — RequestCI&lt;br&gt;[17] — ReqMaxPoll&lt;br&gt;[18:31] Reserved (0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Response Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>enum8</td>
<td>CompletionCode&lt;br&gt;Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIBUTE, INVALID_FILE_IDENTIFIER }&lt;br&gt;See Table 16 for values.</td>
</tr>
<tr>
<td>1:4</td>
<td>uint32</td>
<td>FileAttributeValue&lt;br&gt;This is a fixed-length return value.&lt;br&gt;See Table 12.</td>
</tr>
</tbody>
</table>
### Table 12 — DfGetFileAttribute Returned Value Definition

<table>
<thead>
<tr>
<th>FileAttributeName</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientZeroLengthOnly</td>
<td>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).</td>
</tr>
<tr>
<td>ChangeIndicator</td>
<td>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.</td>
</tr>
<tr>
<td>RequestMaxPoll</td>
<td>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 DRead command before the data may either truncate or wrap, depending on the File PDR FileCapabilities settings.</td>
</tr>
</tbody>
</table>
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.
### Table 13 — DfSetFileAttribute Command Format

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

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Response Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>enum8</td>
<td>CompletionCode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIBUTE, INVALID_FILE_IDENTIFIER, FILE_OPEN }</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Table 16 for values.</td>
</tr>
</tbody>
</table>
Table 14 — DfSetFileAttribute SUCCESS Value Definition

<table>
<thead>
<tr>
<th>FileAttributeName</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientZeroLengthOnly</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.
### Table 15 — DfHeartbeat Command Format

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Request Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:1</td>
<td>uint16</td>
<td>FileDescriptor</td>
</tr>
<tr>
<td>2:5</td>
<td>uint32</td>
<td>RequesterMaxInterval</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The requested maximum supported NegotiatedInterval in milliseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>Response Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>enum8</td>
<td>CompletionCode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Possible values: { PLDM_BASE_CODES, INVALID_FILE_DESCRIPTOR } See Table 16 for values.</td>
</tr>
<tr>
<td>1:4</td>
<td>uint32</td>
<td>ResponderMaxInterval</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The maximum supported NegotiatedInterval in milliseconds from the responder</td>
</tr>
</tbody>
</table>
## 9.7 Error Completion Codes

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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Returned By</th>
<th>Usage Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Various</td>
<td>PLDM_BASE_CODES</td>
<td>File Host &amp; File Client</td>
<td>See the <a href="#">DSP0240 PLDM Base Specification</a>.</td>
</tr>
<tr>
<td>0x80</td>
<td>INVALID_FILE_DESCRIPTOR</td>
<td>File Host &amp; File Client</td>
<td>Invalid FileDescriptor was provided to one of the following commands: DfRead, DfClose, DfHeartbeat.</td>
</tr>
<tr>
<td>0x81</td>
<td>INVALID_DF_ATTRIBUTE</td>
<td>File Host</td>
<td>Invalid attribute or combinations of attributes was provided to one of the following commands: DfOpen, DfGetFileAttribute, DfSetFileAttribute.</td>
</tr>
<tr>
<td>0x82</td>
<td>ZEROLENGTH_NOT_ALLOWED</td>
<td>File Host</td>
<td>DfClose command ZeroLength option set to one (1) of this file is not allowed. See DfClose.</td>
</tr>
<tr>
<td>0x83</td>
<td>EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED</td>
<td>File Host</td>
<td>Attempted to use DfClose command ZeroLength option set to one (1) without proper ownership. See DfClose.</td>
</tr>
<tr>
<td>0x84</td>
<td>EXCLUSIVE_OWNERSHIP_NOT_ALLOWED</td>
<td>File Host</td>
<td>Requested file is not allowed to be opened exclusively. See DfOpen.</td>
</tr>
<tr>
<td>0x85</td>
<td>EXCLUSIVE_OWNERSHIP_NOT_AVAILABLE</td>
<td>File Host</td>
<td>Requested file temporarily cannot be opened exclusively.</td>
</tr>
<tr>
<td>0x86</td>
<td>INVALID_FILE_IDENTIFIER</td>
<td>File Host</td>
<td>Invalid FileIdentifier was provided to one of the following commands: DfOpen, DfGetFileAttribute, DfSetFileAttribute.</td>
</tr>
<tr>
<td>0x87</td>
<td>DFOpen_DIR_NOT_ALLOWED</td>
<td>File Host</td>
<td>Opening a directory is not allowed. See DfOpen.</td>
</tr>
<tr>
<td>0x88</td>
<td>MAX_NUM_FDS_EXCEEDED</td>
<td>File Host</td>
<td>A File Host has run out of FileDescriptors either for this file or overall. See DfOpen.</td>
</tr>
<tr>
<td>0x89</td>
<td>FILE_OPEN</td>
<td>File Host</td>
<td>Attempted to change a file attribute on a currently opened file. See DfSetFileAttribute.</td>
</tr>
<tr>
<td>0x8A</td>
<td>UNABLE_TO_OPEN_FILE</td>
<td>File Host</td>
<td>The File Host is temporarily unable to open a file. See DfOpen.</td>
</tr>
<tr>
<td>0x8B-0xFF</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
233 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.

243 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. 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.

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

9.8.2 DfRead command details

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.

DfRead File Client request requirements are detailed in Table 17:

<table>
<thead>
<tr>
<th>MultipartReceive TransferOperation</th>
<th>All File Types</th>
<th>Additional Requirements when DfOpenRegFIFO is set to one (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFER_FIRST_PART</td>
<td>The FileOffset may be zero (0) (beginning of file) or a nonzero value representing the file offset.</td>
<td>The FileOffset shall be set to zero (0).</td>
</tr>
<tr>
<td></td>
<td>The initial DataTransferHandle shall be zero (0).</td>
<td>The RequestedSectionLengthBytes shall be equal to or less than the Negotiated Transfer Part Size from the most recent successfully completed NegotiateTransferParameters command.</td>
</tr>
<tr>
<td>XFER_NEXT_PART</td>
<td>Not supported</td>
<td></td>
</tr>
<tr>
<td>XFER_COMPLETE</td>
<td>RequestedSectionOffset shall be zero (0).</td>
<td></td>
</tr>
</tbody>
</table>
DfRead File Host response requirements are detailed in Table 18:

Table 18 — DfRead File Host Response Requirements

<table>
<thead>
<tr>
<th>MultipartReceive TransferOperation</th>
<th>All File Types</th>
<th>Additional SerialTxFIFO File Type Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFER_FIRST_PART</td>
<td></td>
<td>The File Host shall respond with a TransferFlag equal to START_AND_END, as required by DSP0240 when DataLengthBytes is less than or equal to the Negotiated Transfer Part Size.</td>
</tr>
<tr>
<td>XFER_COMPLETE</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

DSP0242
PLDM for File Transfer Specification

Version 1.0.0
Published
Table 19 — DfRead Command to MultipartReceive Command Mapping Format

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

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>DfRead Response Data</th>
<th>MultipartReceive Response Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>enum8</td>
<td></td>
<td>CompletionCode (MultipartReceive command)</td>
</tr>
<tr>
<td>1</td>
<td>enum8</td>
<td></td>
<td>TransferFlag</td>
</tr>
<tr>
<td>2:5</td>
<td>uint32</td>
<td></td>
<td>NextDataTransferHandle</td>
</tr>
<tr>
<td>6:9</td>
<td>uint32</td>
<td></td>
<td>DataLengthBytes</td>
</tr>
<tr>
<td>10:N+9</td>
<td>uint8[N]</td>
<td></td>
<td>Data</td>
</tr>
<tr>
<td>N+10:N+13</td>
<td>uint32</td>
<td></td>
<td>DataIntegrityChecksum</td>
</tr>
</tbody>
</table>
9.9 DfFIFOSend (DSP0240 MultipartSend)

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)

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.

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.

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.

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.

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.

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.
### Table 20 — DfFIFOSend to MultipartSend Command Mapping Format

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

<table>
<thead>
<tr>
<th>Byte</th>
<th>Type</th>
<th>DfFIFOSend Response Data</th>
<th>MultipartSend Response Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>enum8</td>
<td></td>
<td>CompletionCode (MultipartSend command)</td>
</tr>
</tbody>
</table>
## 10 ANNEX A (informative) Change Log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.0</td>
<td>2024-07-29</td>
<td>Initial Release</td>
</tr>
</tbody>
</table>
11 ANNEX B (informative) Sensor Threshold Event Examples

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.

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

Table 21 — Sensor Thresholds and Event Examples

<table>
<thead>
<tr>
<th>FileClassification</th>
<th>FMS</th>
<th>PFS CFS</th>
<th>WH</th>
<th>CH</th>
<th>FH</th>
<th>DFSS</th>
<th>FSMS Event</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>SerialTxFiFO</td>
<td>256</td>
<td>0 1</td>
<td>128</td>
<td>230</td>
<td>256</td>
<td>FiU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SerialTxFiFO</td>
<td>256</td>
<td>1 1</td>
<td>128</td>
<td>230</td>
<td>256</td>
<td>FhNC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SerialTxFiFO</td>
<td>256</td>
<td>1 1</td>
<td>128</td>
<td>230</td>
<td>256</td>
<td>FhNC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SerialTxFiFO</td>
<td>256</td>
<td>1 200</td>
<td>128</td>
<td>230</td>
<td>256</td>
<td>FhNC</td>
<td>WH</td>
<td></td>
</tr>
<tr>
<td>SerialTxFiFO</td>
<td>256</td>
<td>256 256</td>
<td>128</td>
<td>230</td>
<td>256</td>
<td>FdwH</td>
<td>FH</td>
<td></td>
</tr>
<tr>
<td>CrashDumpFile</td>
<td>none</td>
<td>1024</td>
<td></td>
<td></td>
<td></td>
<td>PDR RU</td>
<td></td>
<td>Very expensive</td>
</tr>
<tr>
<td>CrashDumpFile</td>
<td>1024</td>
<td>0 1024</td>
<td>512</td>
<td>920</td>
<td>1024</td>
<td>FiaMS</td>
<td>CH</td>
<td></td>
</tr>
<tr>
<td>CrashDumpFile</td>
<td>1024</td>
<td>1024 0</td>
<td>512</td>
<td>920</td>
<td>1024</td>
<td>FiU</td>
<td></td>
<td>DiffClose ZeroLength=1</td>
</tr>
<tr>
<td>FRUDataFile OtherFile</td>
<td>1024</td>
<td>1024 1024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>existing Static File</td>
</tr>
<tr>
<td>FRUDataFile OtherFile</td>
<td>1024</td>
<td>1024 1024</td>
<td></td>
<td></td>
<td></td>
<td>FiU</td>
<td></td>
<td>existing Fixed Length File with update</td>
</tr>
<tr>
<td>BootLog DiagnosticLog SecurityLog OtherLog</td>
<td>1024</td>
<td>100 101</td>
<td>512</td>
<td>920</td>
<td>1024</td>
<td>FiU</td>
<td></td>
<td>Crossed WH</td>
</tr>
</tbody>
</table>
### FileClassification

<table>
<thead>
<tr>
<th>Classification</th>
<th>FMS</th>
<th>PFS/CFS</th>
<th>WH</th>
<th>CH</th>
<th>FH</th>
<th>DFSS</th>
<th>Event</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>BootLog</td>
<td>1024</td>
<td>101</td>
<td>512</td>
<td>920</td>
<td>1024</td>
<td>FiU</td>
<td>WH</td>
<td></td>
</tr>
<tr>
<td>DiagnosticLog</td>
<td></td>
<td>513</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecurityLog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OtherLog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classification</th>
<th>FMS</th>
<th>PFS/CFS</th>
<th>WH</th>
<th>CH</th>
<th>FH</th>
<th>DFSS</th>
<th>Event</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank Row</td>
<td>1111</td>
<td>2222</td>
<td>5555</td>
<td>7777</td>
<td>9999</td>
<td>DFSS</td>
<td>FSMS</td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>3333</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
12 ANNEX C (informative) File PDR FileClassification FileCapabilities Examples

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

<table>
<thead>
<tr>
<th>FileClassification</th>
<th>FileCapabilities</th>
<th>DfOpen Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SerialTxFIFO</td>
<td>ExReadOpen=0</td>
<td>DfOpenReadWrite=Read</td>
<td>File supports polled or pushed read SerialTxFIFO access.</td>
</tr>
<tr>
<td></td>
<td>FileTrunc=1 <em>(Truncate)</em></td>
<td>DfOpenRegFIFO=FIFO</td>
<td>The DfOpen command is for Polled streaming read of the SerialTxFIFO using the DfRead / Multipart Receive command.</td>
</tr>
<tr>
<td></td>
<td>DataType=1 <em>(FIFO)</em></td>
<td>DfOpenPolledPushed=Polled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Polled=1 <em>(Polled)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pushed=1 <em>(Pushed)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DataVolatility=0 <em>(Volatile)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FileModify=0 <em>(Append)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FcZeroLengthPermitted=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FcWritesPermitted=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CrashDumpFile</td>
<td>ExReadOpen=1</td>
<td>DfOpenReadWrite=Read</td>
<td>File supports <em>Exclusive</em> or <em>Non-exclusive</em> read access.</td>
</tr>
<tr>
<td></td>
<td>FileTrunc=1 <em>(Truncate)</em></td>
<td>DfOpenExclusive=Non-exclusive</td>
<td>The DfOpen command is for <em>Regular</em> Non-exclusive reads using the DfRead / Multipart Receive command.</td>
</tr>
<tr>
<td></td>
<td>DataType=0 <em>(Regular)</em></td>
<td>DfOpenRegFIFO=Regular</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Polled=1 <em>(Polled)</em></td>
<td>DfOpenPolledPushed=Polled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pushed=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DataVolatility=1 <em>(Non-volatile)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FileModify=0 <em>(Append)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FcZeroLengthPermitted=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FcWritesPermitted=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CrashDumpFile</td>
<td>ExReadOpen=1</td>
<td>DfOpenReadWrite=Read</td>
<td>File supports <em>Exclusive</em> or <em>Non-exclusive</em> read access.</td>
</tr>
<tr>
<td></td>
<td>FileTrunc=1 <em>(Truncate)</em></td>
<td>DfOpenExclusive=Exclusive</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>DataType=0 <em>(Regular)</em></td>
<td>DfOpenRegFIFO=Regular</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Polled=1 <em>(Polled)</em></td>
<td>DfOpenPolledPushed=Polled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pushed=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DataVolatility=1 <em>(Non-Volatile)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FileModify=0 <em>(Append)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FcZeroLengthPermitted=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FcWritesPermitted=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FileClassification</td>
<td>FileCapabilities</td>
<td>DfOpen Attributes</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BootLog</td>
<td>ExReadOpen=0</td>
<td>DfOpenReadWrite=Read</td>
<td>File supports only non-exclusive read access.</td>
</tr>
<tr>
<td>DiagnosticLog</td>
<td>FileTrunc=0 (Truncate)</td>
<td>DfOpenExclusive=Non-exclusive</td>
<td>The DfOpen command is for non-exclusive reads using the DfRead / Multipart</td>
</tr>
<tr>
<td>SecurityLog</td>
<td>DataType=0 (Regular)</td>
<td>DfOpenRegFIFO=Regular</td>
<td>Receive command with no option to use DfClose command with the ZeroLength=1</td>
</tr>
<tr>
<td>FRUDataFile</td>
<td>Polled=1 (Polled)</td>
<td>DfOpenPolledPushed=Polled</td>
<td>option.</td>
</tr>
<tr>
<td>OtherLog</td>
<td>Pushed=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OtherFile</td>
<td>DataVolatility=1 (Non-volatile)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FileModify=0 (Append)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FcZeroLengthPermitted=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FcWritesPermitted=0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 flows involving File Transfer commands.

1. Initialization example
2. Regular log file read
3. Polled Serial log read
4. Pushed Serial log read

13.1 PLDM for File Transfer initialization example
Figure 4 shows an example of the PLDM for File Transfer initialization sequence.
Example PLDM for File Transfer Initialization

1. GetPLDMVersion TBD
2. GetPLDMVersion response Version=1.1.0
3. PLDM GetPLDMTypes (DSP0240 0x04) TBD
4. PLDM GetPLDMTypes response PLDMTypes=10000101b
Type 0 is base
Type 2 is required for File IO sensors
Type 7 is required for File IO

5. PLDM GetPLDMCommands (DSP0240 0x05)
6. PLDM GetPLDMCommands response

NegotiateTransferParameters Request DSP0240
7. RequesterPartSize=0x200
   RequesterProtocolSupport=Multipart transfer[PLDM for File IO]

NegotiateTransferParameters response
8. ResponderPartSize=0x100
   ResponderProtocolSupport=Multipart transfer[PLDM for File IO]

9. GetPLDMCommands (0x05) PLDMType=0x07
10. GetPLDMCommands response PLDMCommands=0x999

11. GetPDR (DSP0248)
Return the applicable FileIdentifier, entityType, containerID,
   entityInstanceNumber, FileClassification, OemFileClassification,
   FileCapabilities

DSP0240
- FileIO - makes MultipartReceive (0x09) (Req)
- FileIO - makes MultipartSend (0x08) (Req if AsyncIO is supported)
Note: expected only done once on reset / enumeration
NegotiateTransferParameters (0x07) (Req)

Size is MCTP bus specific
Figure 4 — PLDM for File Transfer Initialization Example
### 13.2 Regular log file read

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.
Figure 5 — Regular Log File Read Example - Page 1
MultipartReceive Request
TransferOperation= XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle= 0x00000000
RequestSectionOffset= 0x200 (client calculated)
RequestedSectionLengthBytes= 0x80 (remaining bytes)

MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag= ACKNOWLEDGE_COMPLETION
NextDataTransferHandle= 0x00000200

MultipartReceive Request (DSP0240)
TransferOperation= XFER_FIRST_PART
TransferContext= FileDescriptor (from DfOpen)
DataTransferHandle= 0x00000200
RequestSectionOffset= 0x200 (FileOffset)
RequestedSectionLengthBytes= 0x80 (Section size)

MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag= START_AND_END (only 1 part)
NextDataTransferHandle= 0x00000000 (end of section)
DataLengthBytes= 0x80

MultipartReceive Request
TransferOperation= XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle= 0x00000000
RequestSectionOffset= 0 (no further requests)

MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag= ACKNOWLEDGE_COMPLETION
NextDataTransferHandle= 0x00000000

File Client retains file offset it wants to later read from end of file

File Host:
1) Adds 0x300 bytes to the file,
2) updates the Device File Numeric Sensor / Nominal Value
3) sets the Device State Sensor Update bit to cause an event to be generated to the previously registered FileClient.
**Figure 7 — Regular Log File Read Example - Page 3**
Figure 8 — Regular Log File Read Example - Page 4
13.3 Polled Serial Log Read

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

Figure 9 — Polled Serial Read Example - Page 1
Figure 10 — Polled Serial Read Example - Page 2
MultipartReceive Request (DSP0240)
TransferOperation=XFER_FIRST_PART
TransferContext=FileDescriptor
DataTransferHandle=0x00000000
RequestSectionOffset=0x
RequestedSectionLengthBytes=0x100 (from negotiated value)

MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=START_AND_END
NextDataTransferHandle=0x0
DataLengthBytes=0x40

MultipartReceive Request
TransferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x0
RequestSectionOffset=0x0

MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=ACKNOWLEDGE_COMPLETION
NextDataTransferHandle=0x0

DfClose
FileDescriptor

DfClose response
CompletionCode=SUCCESS

File Client will need to do an additional DfOpen if it wants to read additional data later, but no longer needs to generate DfHeartbeat commands.
13.4 Pushed Serial Log Read

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

---

**Pushed Serial Read**

1. **DfOpen (DSP0242)**
   - FileIdentifier from PDR repository
   - FileCapabilities - Pushed Serial

2. **DfOpen Response**
   - FileDescriptor
   - File Client must be ready for the File Host to immediately start a MultipartSend after it returns the FileDescriptor

3. **MultipartSend Request (DSP0240)**
   - TransferOperation= START_AND_END
   - TransferContext= FileDescriptor
   - DataTransferHandle=0x00000000
   - NextDataTransferHandle= 0x0
   - SectionOffset=0 (DSP0242)
   - SectionLengthBytes=0 (DSP0242)
   - DataLengthBytes= 0x00000100
   - File Host sends 1st 256 bytes
     - Section 0 Part 0

4. **MultipartSend Response**
   - TransferOperation=XFER_COMPLETE
   - File Host moves read pointer
     - Section 0 Part 0

5. **MultipartSend Request (DSP0240)**
   - TransferOperation= START_AND_END
   - TransferContext= FileDescriptor
   - DataTransferHandle=0x00000000
   - NextDataTransferHandle= 0x0
   - SectionOffset=0 (DSP0242)
   - SectionLengthBytes=0 (DSP0242)
   - DataLengthBytes= 0x00000080
   - File Host sends last 128 bytes
     - Section 1 Part 0

6. **MultipartSend Response**
   - TransferOperation=XFER_COMPLETE
   - File Host moves read pointer
     - Section 1 Part 0

7. **DfHeartbeat**
   - RequesterMaxInterval
   - FileDescriptor

8. **DfHeartbeat response**
   - ResponderMaxInterval
   - CompletionCode=SUCCEESS

---

**Figure 12 — Pushed SerialTxFIFO Log Read Example - Page 1**
File Host has more data (64 bytes) to send

**MultipartSend Request**
- TransferOperation = START_AND_END
- TransferContext = FileDescriptor
- DataTransferHandle = 0x00000000
- NextDataTransferHandle = 0x0
- SectionOffset = 0
- SectionLengthBytes = 0
- DataLengthBytes = 0x00000040

File Host sends new 64 bytes
Section 2 Part 0

**MultipartSend Response**
- TransferOperation = XFER_COMPLETE

File Host moves the read pointer
Section 2 Part 0

File Client no longer wants to receive async serial data

Until the File Client receives the DClose response it must be prepared to respond to a MultipartSend

**DClose**
- FileDescriptor

**DClose response**
- CompletionCode = SUCCESS

Figure 13 — Pushed SerialTxFIFO Log Read Example - Page 2