Document Identifier: DSP0242

Date: 2024-01-24

Version: 0.8.0

Platform Level Data Model (PLDM) for File Transfer Specification

Supersedes: None

Document Class: Normative

Document Status: Work in Progress

Document Language: en-US

CONTENTS

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 the DMTF, see http://www.dmtf.org.

1.1 Acknowledgments

The 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
- Yuval Itkin – NVIDIA Corporation
- Tom Joseph – IBM
- Justin King – IBM
- Eliel Louzoun – Intel Corporation
- Hemal Shah – Broadcom, Inc.
- Bill Scherer – Hewlett Packard Enterprise
- 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 serial FIFO 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 outside the scope of 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, which are identified in Clause 2. 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 / faster adoption. The ISO C Language FILE Library semantics, such as open, read, close, are expected to be familiar to the reader. Additionally to leverage to every extent possible DSP0240 multi-part transfers and existing PLDM capabilities including PLDM sensor based event notifications.

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

The following are outside the 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 DSP0248, *Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification* 1.3.0 http://dmtf.org/sites/default/files/standards/documents/DSP0248_1.3.x.pdf

DMTF DSP0249, *Platform Level Data Model (PLDM) State Set Specification* 1.2.0 http://dmtf.org/sites/default/files/standards/documents/DSP0249_1.2.x.pdf

DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes* 1.4.0 http://www.dmtf.org/standards/published_documents/DSP0245_1.4.x.pdf

DMTF DSP0240 *Platform Level Data Model (PLDM) Base Specification* 1.2.0* http://www.dmtf.org/standards/published_documents/DSP0240_1.2.x.pdf

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

DMTF DSP4004, *DMTF Release Process* 2.3 http://www.dmtf.org/standards/published_documents/DSP4004_2.3.x.pdf

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

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

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards* 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. **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.

NegotiatedInterval The maximum negotiated time interval in milliseconds to be used between commands issued by the File Client. See DfHeartbeat for the 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. **IANA** Internet Assigned Numbers Authority

OEM Original Equipment Manufacturer

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 DSP0240 PLDM Base Specification and DSP0248 PLDM for Monitor 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, Close, Delete commands. As there is no Seek command, 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 PLDM Base Specification Discovery and Multiple Part (Multipart) transfer commands, the DSP0248 PLDM for Monitor and Control Specification Platform Data Records (PDR) which includes File Descriptor, Numeric Sensor, State Sensor and Entity Association Records. There are also DSP0248 PLDM for Monitor and Control commands to interact with the Platform Data Records.

The PLDM Initialization Agent discovers the PLDM for File Transfer support including supported specification version and commands as defined in the DSP0240 PLDM Base Specification. The data model definition for a file and an optional associated directory is represented by the DSP0248 PLDM for Monitor and Control Specification File Descriptor Platform Data Record (PDR) with hierarchy expressed with Entity Association Records. The data model provides static (meta) data in the File Descriptor PDR and dynamic data using numeric and state sensors.

The following lists 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 PLDM for Monitor and Control specification defined Platform Data Record (PDR) Repository.
- The File Host may generate events using the DSP0248 PLDM for Monitor and Control specification PlatformEventMessage command. The File Client may elect to receive events using theDSP0248 PLDM for Monitor and Control specification SetEventReceiver command.
- The File Client issues the DfOpen command, using the *FileIdentifier* from a PLDM File Descriptor 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 the DSP0240 PLDM Base Specification 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 PLDM File Descriptor PDR. The dynamic file metadata, such as file size, can be obtained by reading the PLDM numeric sensor for size. The method to retrieve the PLDM PDR and reading a PLDM numeric sensor are defined in the DSP0248 specification.

8.2 File Transfer

A PLDM requester, typically a platform Baseboard Management Controller, will be 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 will describe the terminology and semantics used by this specification as it relates to the ISO C Language FILE library semantics.

8.3.1 Semantics

- ***File***
  - A file is an entity identified by a *PLDM File Descriptor Platform Data Record (PDR)* and has the *EntityType* set to *DeviceFile*
  - A file is a physical object that consumes storage space. The allocated storage may be volatile or non-volatile
  - The File Descriptor 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 Descriptor 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 a PLDM *Entity Association Record*

- ***Directory***

  - A directory is a logical object that associates files within its hierarchy
  - A directory is an entity identified by a *PLDM File Descriptor Platform Data Record (PDR)* and has the *EntityType* set to *DeviceFileDirectory*
  - The File Descriptor PDR has a field, *FileIdentifier* that is a single unique value representing the *directory name* within the File Host hierarchy
  - The directory shall be a PLDM Container of a PLDM *Entity Association Record (EAR)* PDR that associates files into its hierarchy

- ***FileIdentifier***

  - A unique numeric value, obtained from the *PLDM File Descriptor Platform Data Record (PDR)*, and representing the *file name* or the "directory name* within the File Host hierarchy
  - The *FileIdentifier* is used (instead of a name string) for specific FILE type commands such as DfOpen, DfGetFileAttrib, DfSetFileAttrib. When the *FileIdentifier* represents a *directory name*, there are no corresponding commands defined in this specification version.
  - The *FileIdentifier* and the *File Descriptor PDR* field, *FileName* are part of the same PDR record.

- ***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, DfDelete 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 attributes and classifications. There are also dynamic attributes that may be set by the File Client executing the DfSetFileAttrib command if the File Host supports the command and dynamic attribute. Examples of static file attributes which are normatively defined in the DSP0248 PLDM Monitor and Control Specification File Descriptor PDR are: Exclusive Open, File Truncation / Wrapping, File Data Type.

This specification, in collaboration with the DSP0248 PLDM Monitor and Control specification, is recommending industry conventional file (data) classification to allow the File Client to understand the type of contained file data. The typical file classification usage model is:

- Logs are appended
- Files are fixed length
- Serial FIFO (streaming file) requires special handling as documented

Below are some examples of file (data) classifications:

File Classification	Definition
Boot Log	Typically holds device initialization data (events) but has no additional entries after initialization completes
Serial FIFO	Typically removes the data after successful transfer to the receiver or if the FIFO overflows
Diagnostic Log	Typically a variable length file where data can be appended until maximum storage limit is reached.
Crash Dump File	A fixed length file (instance), written one time per crash event, typically containing diagnostic data
FRU Data File	A fixed length file that stores Field Replaceable Unit (FRU) data typically found on add-in adapters
Other (OEM) Log	A file classification that implies growth (appends) for new event (data).
Other (OEM) File	A file classification that implies a "write data once" with no growth after event (data) written.

8.3.3 File and Directory Discovery

Files and Directories are discovered by collecting the PLDM File Descriptor Platform Data Records (PDR) with *EntityType* set to *DeviceFile* or *DeviceFileDirectory*. The 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 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 PLDM Numeric Sensor representing the current file size or the File Client may enable PLDM Events for the file sensors to be alerted when a file has changed. The file does not have to be open for this activity as this is normative PLDM Monitor and Control supporting functionality that is foundational to this specification.

8.3.4 File System Hierarchy Discovery

The file system hierarchy of a File Host is learned through the PLDM Entity Association Record (EAR) Platform Data Records (PDR), defined in DSP0248 PLDM for Monitor and Control. If the File Host implements a hierarchy of directories to contain files, then the File Host shall implement the directory structure using the PLDM 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 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.

8.4 DSP0248 PLDM for Monitor 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 Data Records (PDRs) and specific value assignments within the data model. The reader should consider the foundational PLDM specifications for objects and fields not explicitly stated in this specification.

8.4.1 The File Descriptor Data Model

- The File Descriptor Platform Data Record (Referencing DSP0248 PLDM for Monitor and Control Specification)
  - Every File shall have a File Descriptor PDR that provides static meta data such as the (file) object maximum size.
  - Every File shall have a *EntityType* set to *DeviceFile*
  - Every File shall have a current File Size Monitoring Sensor association
  - Every File shall have a current File State Monitoring Sensor association
- The File Size Monitoring Sensor
  - The File Size Monitoring Sensor shall be implemented as a Compact or Numeric sensor PDR used to report the current file size in bytes 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 an explicit DSP0248 PLDM for Monitor and Control SetNumericSensorEnable command
  - The File Size Monitoring Sensor shall match the monitored File Descriptor 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)
  - The File Size Monitoring Sensor *RateUnit* shall be set to *None*
  - Every File Size Monitoring Sensor shall have a *WarningLow*, *WarningHigh*, *CriticalHigh*, *FatalHigh* whose initial values are set by the File Host
  - The File Size monitoring Sensor should be supported by the DSP0248 PLDM for Monitor and Control GetSensorThresholds command and SetSensorThresholds command to allow a file client to receive events as a file is modified or deleted.

- For the associated File Descriptor PDR, the threshold values are defined as:

  - WarningLow: Initially set by the File Host to zero (0) to allow detection of zero length file delete operation, when the File Host sets the file size to zero (0). The File Client should not adjust this value.
  - WarningHigh: Initially set by the File Host to indicate 50% capacity status. The File Client may adjust this value lower or higher using the DSP0248 PLDM Monitor and Control command, SetSensorThresholds, as a method to generate an event when the file size changes
  - CriticalHigh: Initially set by the File Host to indicate 90% capacity status. The File Client may adjust this value lower or higher using the DSP0248 PLDM Monitor and Control command, SetSensorThresholds, as a method to generate an event when the file changes. This specification does not recommend setting the CriticalHigh greater than what the File Hosts initially sets.
  - FatalHigh: Initially set by the File Host to indicate data has been lost. The File Client shall not adjust this value.

- The File State Monitoring Sensor

  - The File State Monitoring 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). Optionally, the File State Monitoring Sensor may be used to generate PLDM events or an explicit DSP0248 PLDM for Monitor and Control SetStateSensorEnables command
  - The File State Monitoring Sensor shall match the monitored File Descriptor PDR *EntityType*, *EntityInstance* and *ContainerID* fields to establish its association to the monitored file
  - The File State Monitoring Sensor shall implement the DSP0249 PLDM State Set Specification State Set *Device File (68)*
  - The File Host shall update the the File State Monitoring Sensor after every GetStateSensorReadings command, typically switching the state between *File Data is Reported Modified* and *No File Data Modification Reported* states, unless another file event is present which remain persistent until the event trigger is cleared.

8.4.2 The Directory Descriptor Data Model

- The File Descriptor Platform Data Record (Referencing DSP0248)
    - Every directory shall have a File Descriptor PDR (DSP0248)
    - Every directory shall have a *EntityType* set to *DeviceFileDirectory*
    - The File Descriptor PDR field, *FileAttributes* shall be set to zero (0)
    - The File Descriptor PDR field, *FileVersion* shall be set to unversioned (0xFFFFFFFF)
    - The File Descriptor PDR field, *FileMaximumSize* shall be set to special value 0xFFFFFFFF
- A directory shall be represented as a PLDM Entity Association Record PDR, such that the directory *ContainerID* shall be the directory EAR PDR *ContainerID*
- The PLDM Entity Association Record PDR *AssociationType* shall always be set to *LogicalContainment*
- The PLDM Entity Association Record PDR fields: *ContainerEntityType* and *ContainerEntityInstanceNumber* shall match the associated directory File Descriptor PDR *EntityType* and *EntityInstance* values
- The PLDM Entity Association Record PDR 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 Descriptor PDR *EntityType*, *EntityInstance* and *ContainerID* fields listed in the directory Entity Association Record PDR *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*

8.4.3 File Data Model

In Figure 1 the numeric and state sensors associated with the file have matching *EntityType*, *EntityInstance* and *ContainerID* fields to the File Descriptor PDR.

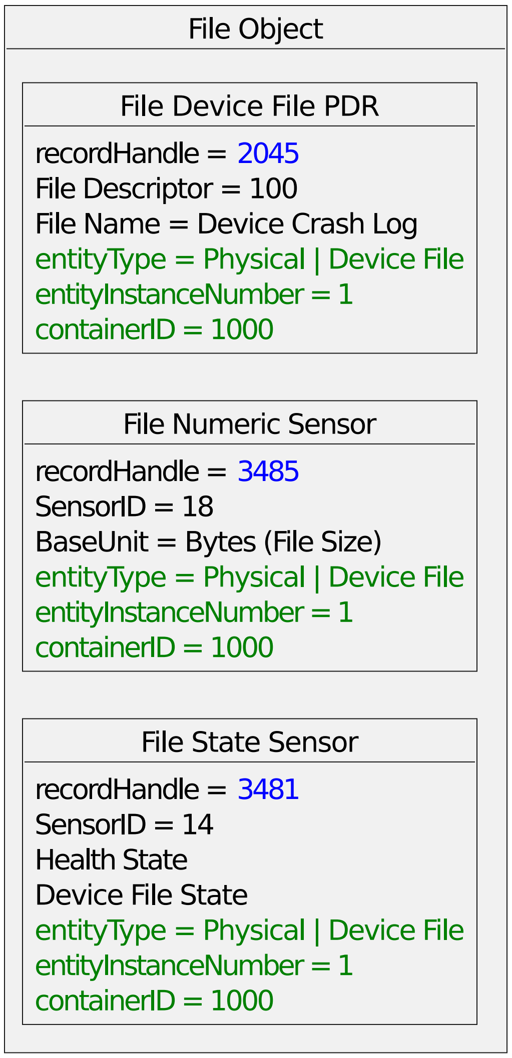


Figure 1 — PLDM for File Transfer File Data Model implicit association

9 PLDM for File Transfer Commands

This section describes the commands that shall be used for File Transfer. Table 1 consists of the codes assigned to commands. These commands have their own PLDM message type, which is defined in DSP0245.

Table 1 — PLDM for File Transfer Command Codes

Command	Code Value	File Host support	File Client support
DfOpen	0x01	Required	Required
DfClose	0x02	Required	Optional
DfDelete	0x03	Optional	Optional
DfGetFileAttrib	0x04	Optional	Optional
DfSetFileAttrib	0x05	Optional	Optional
DfHeartbeat	0x06	Optional	Conditional
Reserved	0x07-0xFF		
DfRead	MP	Required	Required
[DfFifoSend]	MP	Optional	Optional

NOTE:
MP = DSP0240 PLDM Multipart Transfer

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

9.1 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 2, uses a File Descriptor PDR *FileIdentifier* field to access the specific file.

The returned *FileDescriptor* is used by the File Client for subsequent DfRead, DfClose, DfDelete commands, including the DfHeartbeat command. The unique *FileDescriptor* can optionally be used by the File Host to track how many File Clients have a file open.

The DfOpen command only supports the File Descriptor PDR with the entity type set to *Device File*. If the File Host receives a DfOpen command with a *FileIdentifier* associated to a File Descriptor PDR with the entity type set to *Device File Directory*, the File Host shall return a DFOPEN_DIR_NOT_ALLOWED *CompletionCode*.

If the File Client sets the *DfOpenRegFifo* attribute to one (1) and *DfOpenPolledAsync* attribute to one (1) 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.

File Client / File Host DfOpenAttrib rules:

- If bit *DfOpenRegFifo* is zero (0), then the bit *DfOpenPolledAsync* shall be ignored by the File Host and shall be zero (0)

If the File Client issues a DfOpen command with an invalid *FileIdentifier*, then the File Host shall return a INVALID_FILE_IDENTIFIER *CompletionCode*.

If the File Client issues a DfOpen command with an invalid combination of *DfOpenAttrib* or unsupported *DfOpenAttrib* for the requested *FileIdentifier* then the File Host shall return the INVALID_DFOPEN_ATTRIB *CompletionCode*.

If the File Host can not establish exclusive ownership of the requested *FileIdentifier* at this time or the File Client requested *FileIdentifier* does not support exclusive ownership, then the File Host shall return the EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED *CompletionCode* to the File Client issued DfOpen command.

If the File Client successfully completes a DfOpen command with the *DfOpenAttrib DfOpenExclusive* set to one (1), the File Host shall not make any updates, including changing the length of the file represented by the request *FileDescriptor*. If the File Host can not support this requirement then it shall return the EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED *CompletionCode*.

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

- The File Client would like to issue a DfDelete command on this file
- The File Client needs to make sure the file does not change while it is reading the file.

Due to the restrictions placed on the File Host with a file opened exclusively, the File Client should minimize the time the file is opened with the *DfOpenAttrib DfOpenExclusive* set to one (1).

Table 2 — DfOpen command format

Byte	Type	Request Data
0	uint16	**FileIdentifier**
2..3	bitfield16	**DfOpenAttrib** [0] DfOpenExclusive — Non-exclusive Read (0) / Exclusive Read (1) [1] DfOpenRegFifo — regular (0) or FIFO (serial) (1) [2] DfOpenPolledAsync — Polled (0) or Pushed / Async (DfFifoSend) (1) [3..15] Reserved (0)

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_FILE_IDENTIFIER, INVALID_DF_ATTRIB, EXCLUSIVE_OWNERSHIP_NOT_ALLOWED, EXCLUSIVE_OWNERSHIP_NOT_AVAILABLE, EXCLUSIVE_OWNERSHIP_REQUIRED, DFOPEN_DIR_NOT_ALLOWED, MAX_NUM_FDS_EXCEEDED } See Table 10 for values
1..2	uint16	**FileDescriptor**

9.2 DfClose Command

The DfClose command, as described in Table 3, is used to by the File Client to tell a File Host it no longer needs access to a file. After the File Client has successfully completed a DfClose, it no longer needs to issue DfHeartbeat commands for that file.

If the File Client issues a DfClose command with an invalid *FileDescriptor*, then the File Host shall return an INVALID_FILE_DESCRIPTOR *CompletionCode*.

If there are no other outstanding DfOpen, then a File Client may open a file for exclusive access in preparation of a optional DfDelete command.

Table 3 — DfClose command format

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

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: {PLDM_BASE_CODES, INVALID_FILE_DESCRIPTOR} See Table 10 for values

9.3 DfDelete Command

The DfDelete command, as described in Table 4, is used to by the File Client to request a File Host delete the file identified by the *FileDescriptor*.

The File Host may implement the DfDelete command in one of two ways. The File Client shall support both DfDelete implementations

9.3.1 - Delete Specific File Descriptor PDR

This method is an expensive operation since multiple things must change on the host such as the GetPDRRepositoryInfo *UpdateTime*, the GetPDRRepositorySignature *PdrRepositorySignature* and additionally, send a *PldmPDRRepositoryChgEvent* for both the File Descriptor PDR deletion and subsequent File Descriptor PDR addition. This is not considered to be the best practice.

9.3.2 - Set the File Size Sensor to Zero (0)

This method is simple as it changes a value that already may have a lower threshold value set which can trigger a DSP0248 PLDM Sensor Event and also can be detected by a PLDM GetSensorReading command.

9.3.3 Implementing the DfDelete Command

The File Host shall support the DfDelete command if any of the files in the file PLDM PDR Repository has the PDR FileAttributes *FcDeletePermitted* bit set to one (1).

A File Client shall successfully complete a DfOpen command with the DfOpenExclusive bit equal to one (1) prior to issuing a DfDelete command.

If a File Host supports the DfDelete command and the *FileDescriptor* represents a file that has the FileAttributes FcDeletePermitted bit set to one (1) and the File Host can not delete the file at the time of the request, the *CompletionCode*, DFDELETE_NOT_ALLOWED, is returned.

If the File Client has not successfully completed a DfOpen command with the DfOpenExclusive bit equal to one (1), then the File Host shall return a EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED *CompletionCode*.

If the *FileDescriptor* provided by the File Client is or is no longer valid, the File Host shall return a INVALID_FILE_DESCRIPTOR *CompletionCode*.

Upon successful completion of the DfDelete command, the equivalent of a DfClose command shall be completed and the *FileDescriptor* referenced by the DfDelete shall no longer be valid.

Table 4 — DfDelete command format

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

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_FILE_DESCRIPTOR, DFDELETE_NOT_ALLOWED, EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED } See Table 10 for values

9.4 DfGetFileAttrib Command

The DfGetFileAttrib command, as described in Table 5, is used to by the File Client to obtain attributes of a file.

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

If the File Host does support the requested AttribReq for that *FileDescriptor* then it shall return the INVALID_DF_ATTRIB *CompletionCode*.

Table 5 — DfGetFileAttrib command format

Byte	Type	Request Data
0..1	uint16	**FileIdentifier** This is the *FileIdentifier* returned in the PLDM File Descriptor PDR for this file (directory)
2..3	bitfield32	**FileAttribReq** [0] ClientDeleteOnly — Request ClientDeleteOnly attribute (1) [1..15] Reserved (0) [16] RequestCI — Request Change Indictor (1) [17] ReqMaxPoll — Request maximum poll interval (1) [18..31] Reserved (0)

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIB, INVALID_FILE_IDENTIFIER} See Table 10 for values
1	uint8	**FileAttribReturnCount** This is the number of AttribReq values returned in this response. The response values are returned in the bit order requested
Var	uint32	**FileAttribValue** This will be a fixed length return value, in FileAttribReq bit-wise order. See Table 6

Table 6 — DfGetFileAttrib Returned Value Definition

FileAttribName	Definition
ClientDeleteOnly	This has been designed by the File Client to be preserved until explicitly deleted by the File Client. The file should be opened exclusively if this attribute is set
RequestCI	Returned when the FileAttribReq *RequestCI* bit is set. The *ChangeIndicator* file attribute is generated by the File Host either at the time of the reception of the *DfGetFileAttrib* or at the time when the file was last changed. The File Client can 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. It is recommended *ChangeIndicator* be a 32-bit CRC.
RequestMaxPoll	This is the maximum time allowed between reading the PLDM file size sensor or DfRead command before the data may either truncate or wrap, depending on the PLDM File Descriptor PDR *FileAttribute* settings.

9.5 DfSetFileAttrib Command

The DfSetFileAttrib command, as described in Table 7, is used to by the File Client to set specific file attributes such as file preservation.

If the File Host does support the requested *AttribReq* for that *FileDescriptor* then it shall return the INVALID_DF_ATTRIB *CompletionCode*.

Table 7 — DfSetFileAttrib command format

Byte	Type	Request Data
0..1	uint16	**FileIdentifier** This is the *FileIdentifier* returned in the PLDM File Descriptor PDR for this file (directory)
2..3	bitfield16	**FileAttribSet** [0] ClientDeleteOnly (1) — Set the *ClientDeleteOnly* attribute (1) [1..15] Reserved for future(0)
1	uint8	**FileAttribSetCount** This is the number of *FileAttribValue* set in this response. The set values are sent in the ordinal *FileAttribSet* bit order. This value shall be in range [1..16]
Var	uint32	**FileAttribValue** This will be a fixed length value, in *FileAttribSet* bit-wise order. See Table 8

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_DF_ATTRIB, INVALID_FILE_IDENTIFIER, FILE_OPEN} See Table 10 for values
1..2	bitfield16	**AcceptedFileAttribSet** This field shall have the corresponding bit set for a *FileAttribSet* attribute change accepted. If the *CompletionCode* returned is SUCCESS, then this value shall be equal to request parameter *FileAttribSet*. If the *CompletionCode* is not equal to SUCCESS, this field shall set the accepted bits from the request *FileAttribSet* parameter. The File Client shall execute a Logical Exclusive OR (XOR) of the request *FileAttribSet* parameter and the response *AcceptedFileAttribSet* parameter to learn which attributes were accepted or rejected. When the File Host rejects a request *FileAttribSet*, the returned *CompletionCode* shall be INVALID_DF_ATTRIB

Table 8 — DfSetFileAttrib Returned Value Definition

FileAttribName	Definition
ClientDeleteOnly	If *ClientDeleteOnly* FileAttribValue is set to one (1), then the file has been designated by the File Client to be preserved until explicitly deleted by the File Client. Setting the *ClientDeleteOnly* FileAttribValue to zero(0) allows the file to be deleted by the File Host or by the File Client. The file shall be subsequently opened exclusively if this attribute is set to one. If the file is currently open, the File Host shall not change the ClientDeleteOnly state and return the FILE_OPEN *CompletionCode* There are 2 options to clear this attribute: 1) The File Client can issue a DfSetFileAttrib command with the *ClientDeleteOnly* attribute set to zero (0) or 2) The File Client can open the file exclusively and issue a DfDelete command if the File Host supports deleting this file

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 Serial FIFO type files.

The DfHeartbeat command, as described in Table 9, enables:

- initialization to negotiate the maximum time interval allowed between the last DfOpen, DfRead, DfHeartbeat command and when the File Host may optionally close the 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 will be used to inform the File Client when a Serial FIFO 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 of *RequestorMaxInterval* and *ResponderMaxInterval* is defined as the current *NegotiatedInterval*. The File Client shall issue a DfHeartbeat or DfRead commands using the current *NegotiatedInterval* as the maximum period between DfOpen, DfRead, DfHeartbeat commands.

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

If the *FileDescriptor* requested by the File Client is or is no longer valid, the File Host shall return a INVALID_FILE_DESCRIPTOR *CompletionCode*.

Table 9 — DfHeartbeat command format

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

Byte	Type	Response Data
0	enum8	**CompletionCode** Possible values: { PLDM_BASE_CODES, INVALID_FILE_DESCRIPTOR } See Table 10 for values
1..4	uint32	**ResponderMaxInterval** The maximum supported *NegotiatedInterval* in milliseconds from the responder

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 10. The context in which these are used are described below as well.

Table 10 — PLDM File Transfer Completion Codes

Value	Name	Returned By	Usage Description
Various	PLDM_BASE_CODES	File Host & File Client	Refer DSP0240 PLDM Base Specification
0x80	INVALID_FILE_DESCRIPTOR	File Host & File Client	Invalid *FileDescriptor* was provided to one of the following commands: DfRead, DfClose, DfDelete, DfHeartbeat.
0x81	INVALID_DF_ATTRIB	File Host	Invalid attribute or combinations of attributes was provided to one of the following commands: DfOpen, DfGetFileAttrib, DfSetFileAttrib.
0x82	DFDELETE_NOT_ALLOWED	File Host	Deletion of this file is not allowed DfDelete
0x83	EXCLUSIVE_OWNERSHIP_NOT_ESTABLISHED	File Host	Attempted to delete a file without proper ownership, DfDelete
0x84	EXCLUSIVE_OWNERSHIP_NOT_ALLOWED	File Host	Requested file is not allowed to be opened exclusively DfOpen
0x85	EXCLUSIVE_OWNERSHIP_NOT_AVAILABLE	File Host	Requested file can not be opened exclusively at this time
0x86	INVALID_FILE_IDENTIFIER	File Host	Invalid *FileIdentifier* was provided to one of the following commands: DfOpen, DfGetFileAttrib, DfSetFileAttrib
0x87	EXCLUSIVE_OWNERSHIP_REQUIRED	File Host	Exclusive ownership required for the requested attributes DfOpen
0x88	DFOPEN_DIR_NOT_ALLOWED	File Host	Opening a directory is not allowed DfOpen
0x89	MAX_NUM_FDS_EXCEEDED	File Host	A File Host has run out *FileDescriptor* either for this file or overall DfOpen
0x8A	FILE_OPEN	File Host	Attempted to change a file attribute on a currently opened file DfSetFileAttrib
0x8B-0xFF	Reserved	Reserved	

9.8 DfRead (DSP0240 MultipartReceive)

The DfRead command is a PLDM for File Transfer (type) specific implementation of the DSP0240 PLDM Base Specification *Multipart Transfer Commands*, and specifically the MultipartReceive command. The DSP0240 PLDM Base Specification 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*.

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 the DSP0240 PLDM Base Specification with the mappings defined in Table 12.

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.

Before the first DfRead command is initiated between the File Client and File Host, the DSP0240 PLDM Base Specification NegotiateTransferParameters command shall be initiated by the File Client to the File Host. The File Host shall successfully respond to this request before any MultipartReceive commands are executed by the File Client. The NegotiateTransferParameters command is typically executed once during PLDM initialization and applies to both MultipartReceive and MultipartSend commands.

This specification requires that PLDM for File Transfer PLDM Type is specified in the NegotiateTransferParameters command fields *RequesterProtocolSupport* and *ResponderProtocolSupport*.

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

9.8.1 Implementation Note: Serial FIFO type file characteristics

Files classified as a Serial FIFO have specific characteristics, similar to an endpoint streaming data to a Universal Asynchronous Receiver-Transmitter (UART) device. The following rules apply:

- Seeking not supported (MultipartReceive *Requested SectionOffset* shall be set to zero)
- Single part per section (MultipartReceive *Requested SectionOffset* shall be set to zero)

- Single section (no pigging backing multiple sections as the offset is always zero)
- The File Host shall move the FIFO read pointer when the the File Client sets the MultipartReceive *TransferOperation* field to XFER_COMPLETE
- The File Client shall issue the MultipartReceive to read a Serial FIFO type file until the data length is less than the negotiated part size, indicating all the data has been transferred at this time interval.

The DfRead command is implemented using the *Multipart Transfer Commands*, as Table 11 describes.

Table 11 — PLDM for File Transfer *Multipart Transfer Commands* Mapping

Command	File Host Request	File Client Respond	File Client Request	File Host Respond
NegotiateTransferParameters	Not Supported	Not Supported	Mandatory	Mandatory
MultipartSend	Optional	Optional	Not supported	Not Supported
MultipartReceive	Not Supported	Not Supported	Mandatory	Mandatory

The DfRead command will map to the MultipartReceive command as described in Table 14. 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 12:

Table 12 — DfRead File Client Request Requirements

MultipartReceive TransferOperation	**All File Types**	**Additional Requirements when *DfOpenRegFifo* is set to one (1).**
XFER_FIRST_PART	The *FileOffset* may be zero (0) (beginning of file) or a non-zero value representing the file offset. The initial *DataTransferHandle* shall be zero (0).	The *FileOffset* shall be set to zero (0). The *RequestedSectionLengthBytes* shall be equal to or less than the *Negotiated Transfer Part Size* from the most recent successfully completed NegotiateTransferParameters command.
XFER_NEXT_PART or XFER_COMPLETE		XFER_COMPLETE: *RequestedSectionOffset* shall be zero (0)

DfRead File Host response requirements are detailed in Table 13:

Table 13 — DfRead File Client Request Requirements

MultipartReceive TransferOperation	**All File Types**	**Additional Serial FIFO File Type Requirements**
XFER_FIRST_PART		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*.
XFER_COMPLETE		The File Host shall move the read pointer ahead by the number of bytes successfully transferred. The data is not retained by the File Host.

Table 14 — DfRead command to MultipartReceive command mapping format

Byte	Type	DfRead Request Data	MultipartReceive Request Data
0	uint8	0x07	PLDMType
1	enum8		TransferOperation
2..5	uint32	**FileDescriptor**	TransferContext
6..9	uint32		DataTransferHandle
10..13	uint32	**FileOffset**	RequestedSectionOffset
14..17	uint32		RequestedSectionLengthBytes

Byte	Type	DfRead Response Data	MultipartReceive Response Data
0	enum8		CompletionCode (MultipartReceive command)
1	enum8		TransferFlag
2..5	uint32		NextDataTransferHandle
6..9	uint32		DataLengthBytes
10..N+10	uint8[N]		Data
N+11..N+14	uint32		DataIntegrityChecksum

9.9 DfFifoSend (DSP0240 MultipartSend)

The DfFifoSend command is used exclusively for a file PDR with attribute type *PushedFifo* and are opened by the File Client with the DfOpen command DfOpenAttrib set:

- DfOpenExclusive — Exclusive Read (1)
- DfOpenRegFifo — FIFO (serial) (1)
- DfOpenPolledAsync — Pushed / Async (DfFifoSend) (1)

The DfFifoSend command is equated to a PLDM for File Transfer (type) 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 expect a DfFifoSend command to take place immediately after the File Host successfully responds to the DfOpen command request.

The File Host will asynchronously, without prompting and when file data is placed in the FIFO 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 15:

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 shall be 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 shall move the read pointer ahead by the number of bytes successfully transferred and this data section is no longer re-transmittable (DSP0240 section complete).

Table 15 — DfFifoSend to MultipartSend command mapping format

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

Byte	Type	DfFifoSend Response Data	MultipartSend Response Data
0	enum8		CompletionCode (MultipartSend command)
1	enum8		NextTransferOperation
2..5	uint32		NextDataTransferHandle

9.10 PLDM for File Transfer sensor usage examples

Figure 2 shows a flat file sensor usage example

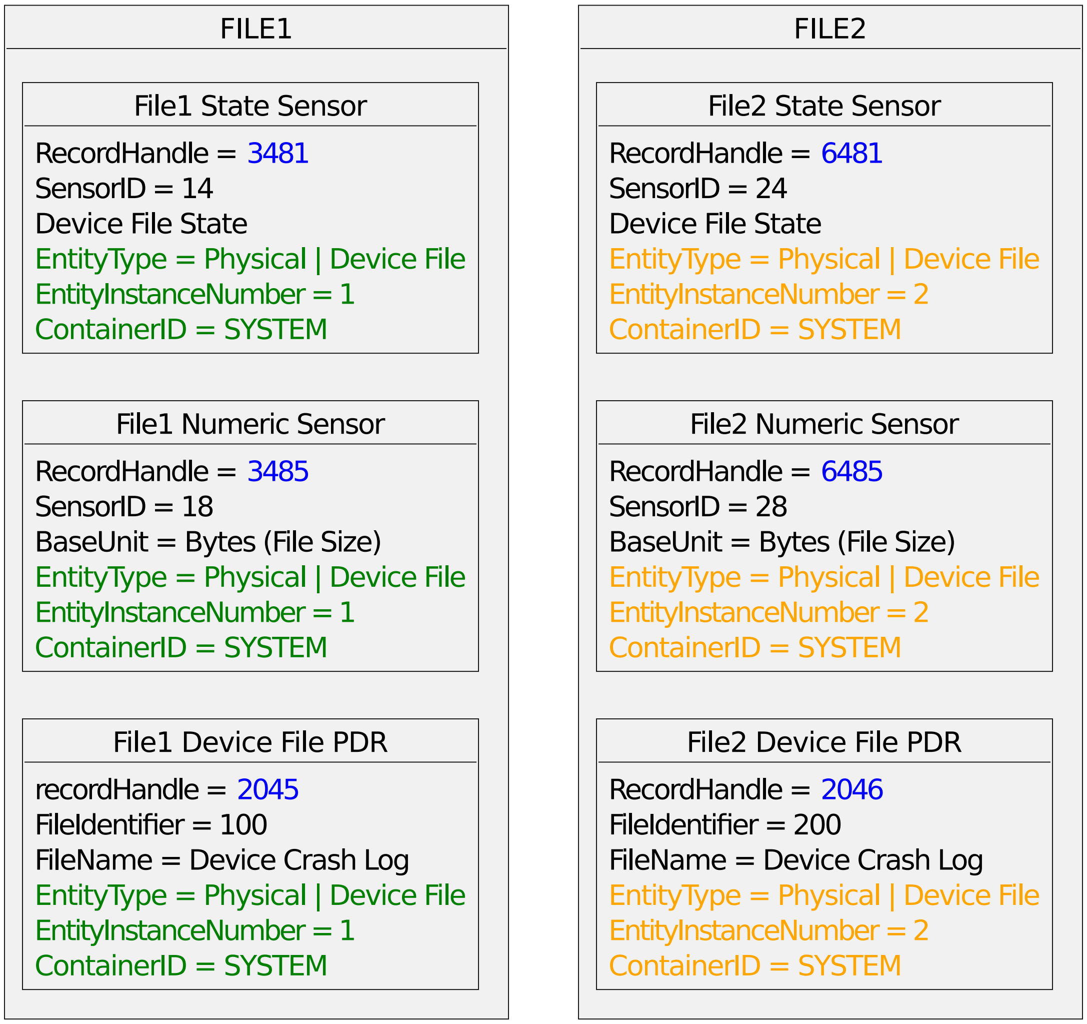


Figure 2 — PLDM for File Transfer Flat File Sensor Usage Example

Figure 3 shows a Directory EAR sensor usage example

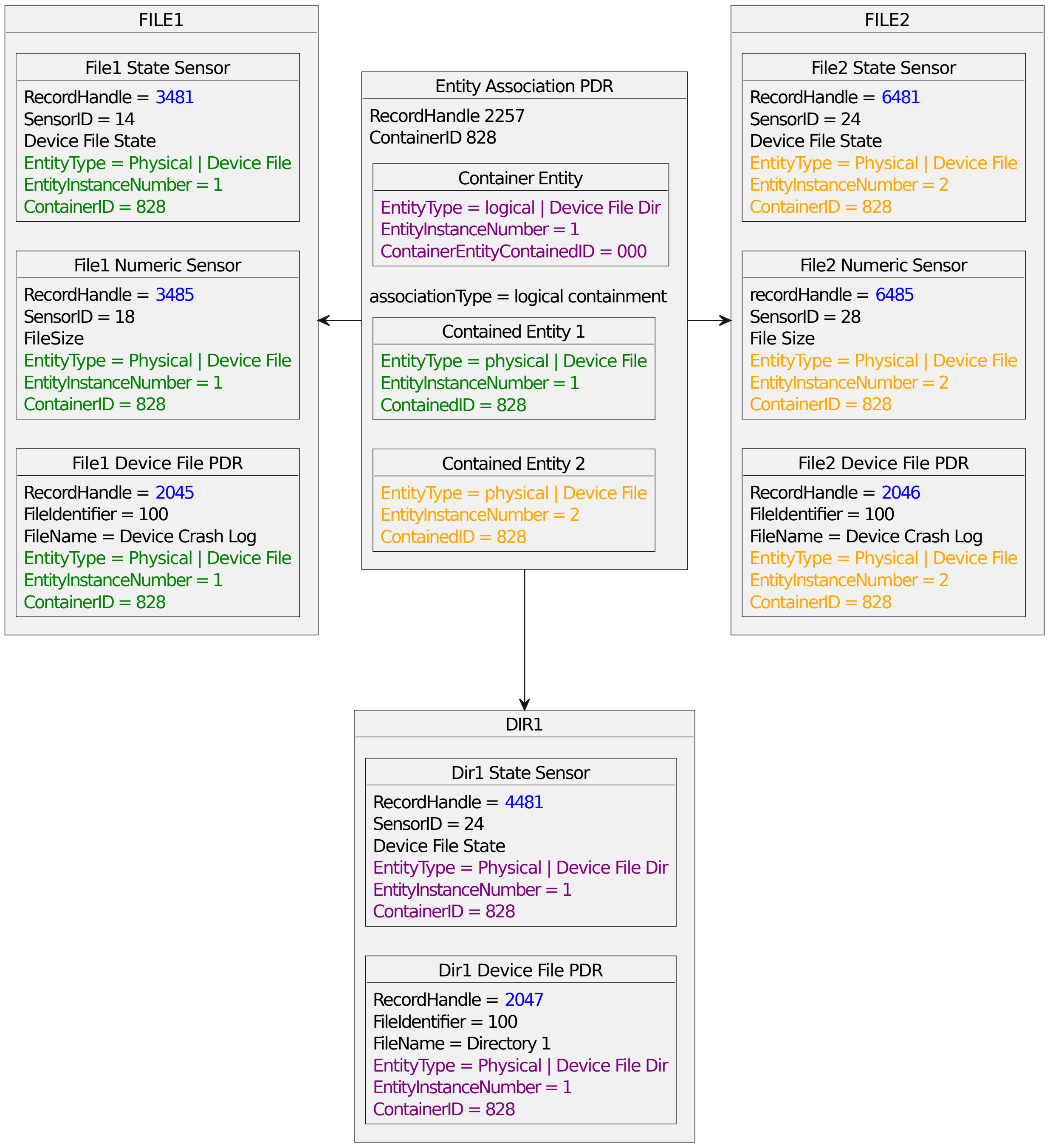


Figure 3 — PLDM for File Transfer Directory EAR Sensor Usage Example

10 PLDM for File Transfer Examples

This informative section describes typical flows involving File Transfer commands.

1. Initialization example
2. Regular file read
3. Polled serial log read
4. Async serial log read

10.1 PLDM for File Transfer initialization example

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

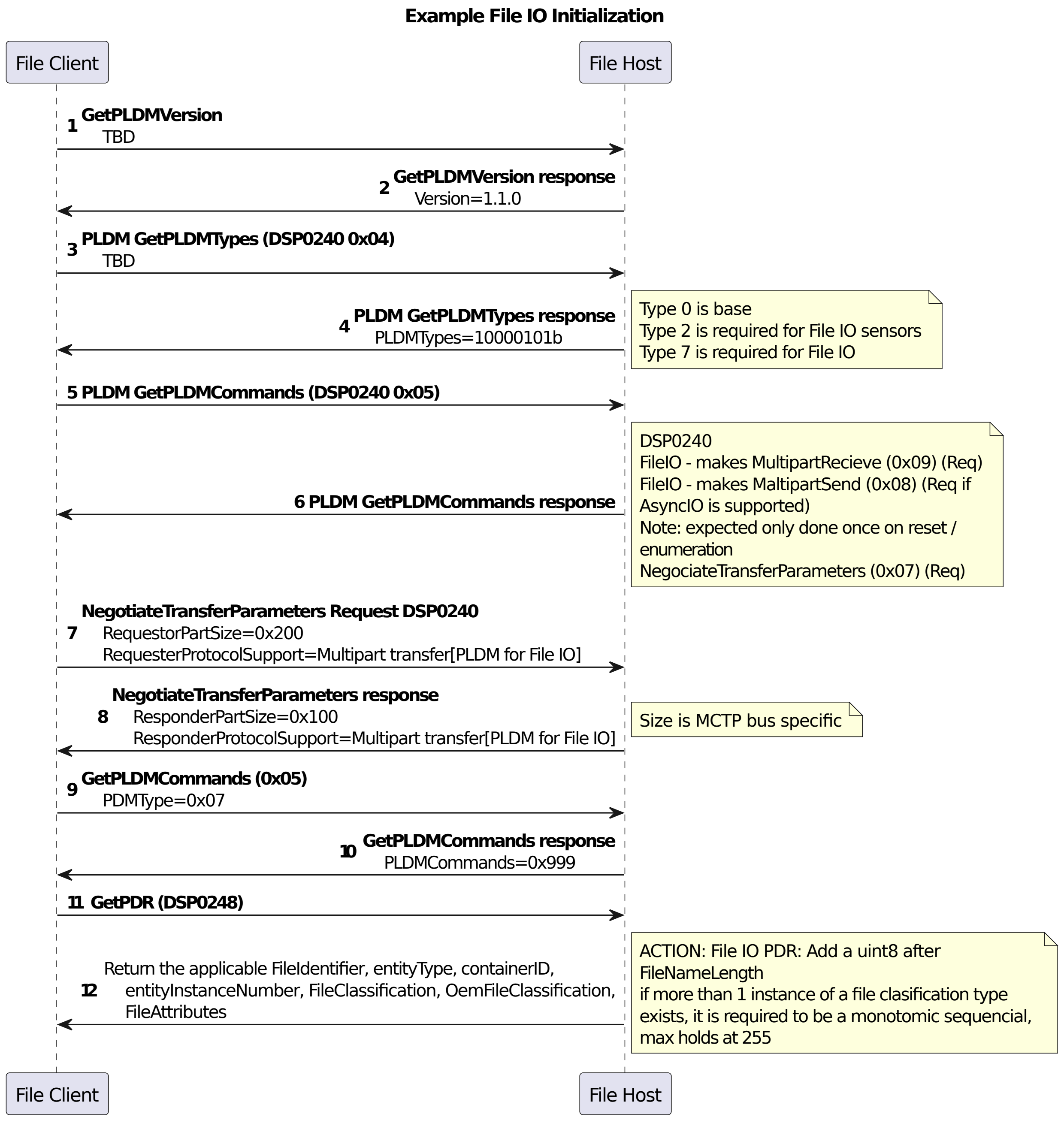

Example File IO Initialization
File Client
File Host
1 GetPLDMVersion
TBD
2 GetPLDMVersion response
Version=1.1.0
3 PLDM GetPLDMTypes (DSP0240 0x04)
TBD
4 PLDM GetPLDMTypes response
PLDMTypes=10000101b
Type 0 is base
Type 2 is required for File IO sensors
Type 7 is required for File IO
5 PLDM GetPLDMCommands (DSP0240 0x05)
6 PLDM GetPLDMCommands response
DSP0240
FileIO - makes MultipartRecieve (0x09) (Req)
FileIO - makes MaltipartSend (0x08) (Req if AsyncIO is supported)
Note: expected only done once on reset / enumeration
NegociateTransferParameters (0x07) (Req)
7 NegotiateTransferParameters Request DSP0240
RequestorPartSize=0x200
RequesterProtocolSupport=Multipart transfer[PLDM for File IO]
8 NegotiateTransferParameters response
ResponderPartSize=0x100
ResponderProtocolSupport=Multipart transfer[PLDM for File IO]
Size is MCTP bus specific
9 GetPLDMCommands (0x05)
PDMType=0x07
10 GetPLDMCommands response
PLDMCommands=0x999
11 GetPDR (DSP0248)
12 Return the applicable FileIdentifier, entityType, containerID, entityInstanceNumber, FileClassification, OemFileClassification, FileAttributes
ACTION: File IO PDR: Add a uint8 after FileNameLength
if more than 1 instance of a file clasification type exists, it is required to be a monotomic sequencial, max holds at 255
File Client
File Host

Figure 4 — PLDM for File Transfer initialization example

10.2 Regular file read

Figures 5, 6, 7, and 8 show an example of a regular file read. The part size of 0x100 (256) bytes is a result of a previously executed NegotiateTransferParameters command. As defined in 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

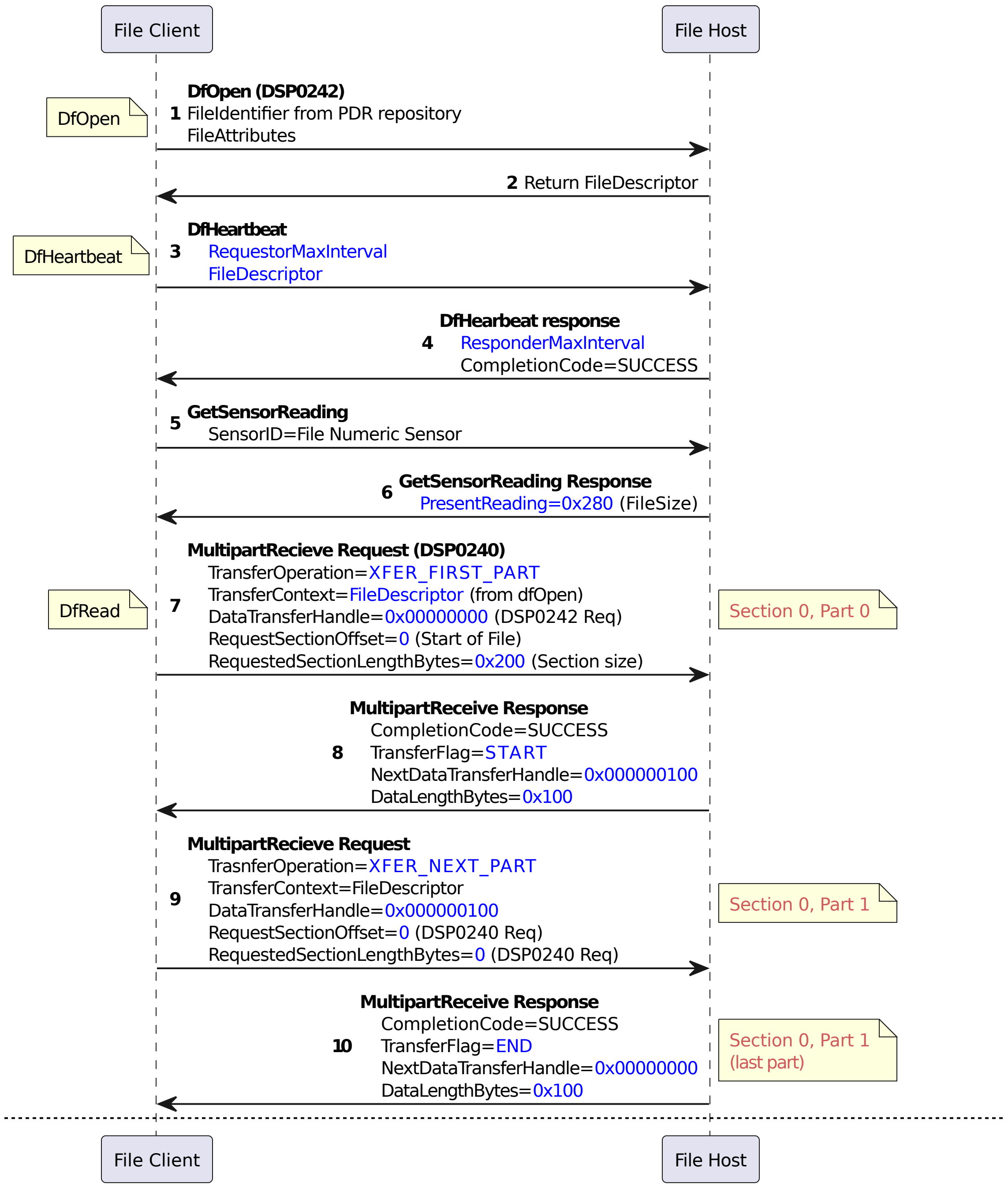

File Client
File Host
DfOpen
DfOpen (DSP0242)
1 FileIdentifier from PDR repository
FileAttributes
2 Return FileDescriptor
DfHeartbeat
DfHeartbeat
3 RequestorMaxInterval
FileDescriptor
DfHearbeat response
4 ResponderMaxInterval
CompletionCode=SUCCESS
5 GetSensorReading
SensorID=File Numeric Sensor
6 GetSensorReading Response
PresentReading=0x280 (FileSize)
DfRead
MultipartRecieve Request (DSP0240)
7 TransferOperation=XFER_FIRST_PART
TransferContext=FileDescriptor (from dfOpen)
DataTransferHandle=0x00000000 (DSP0242 Req)
RequestSectionOffset=0 (Start of File)
RequestedSectionLengthBytes=0x200 (Section size)
Section 0, Part 0
MultipartReceive Response
8 CompletionCode=SUCCESS
TransferFlag=START
NextDataTransferHandle=0x000000100
DataLengthBytes=0x100
MultipartRecieve Request
9 TrasnferOperation=XFER_NEXT_PART
TransferContext=FileDescriptor
DataTransferHandle=0x000000100
RequestSectionOffset=0 (DSP0240 Req)
RequestedSectionLengthBytes=0 (DSP0240 Req)
Section 0, Part 1
MultipartReceive Response
10 CompletionCode=SUCCESS
TransferFlag=END
NextDataTransferHandle=0x00000000
DataLengthBytes=0x100
Section 0, Part 1 (last part)
File Client
File Host

Figure 5 — Regular file read example - Page 1

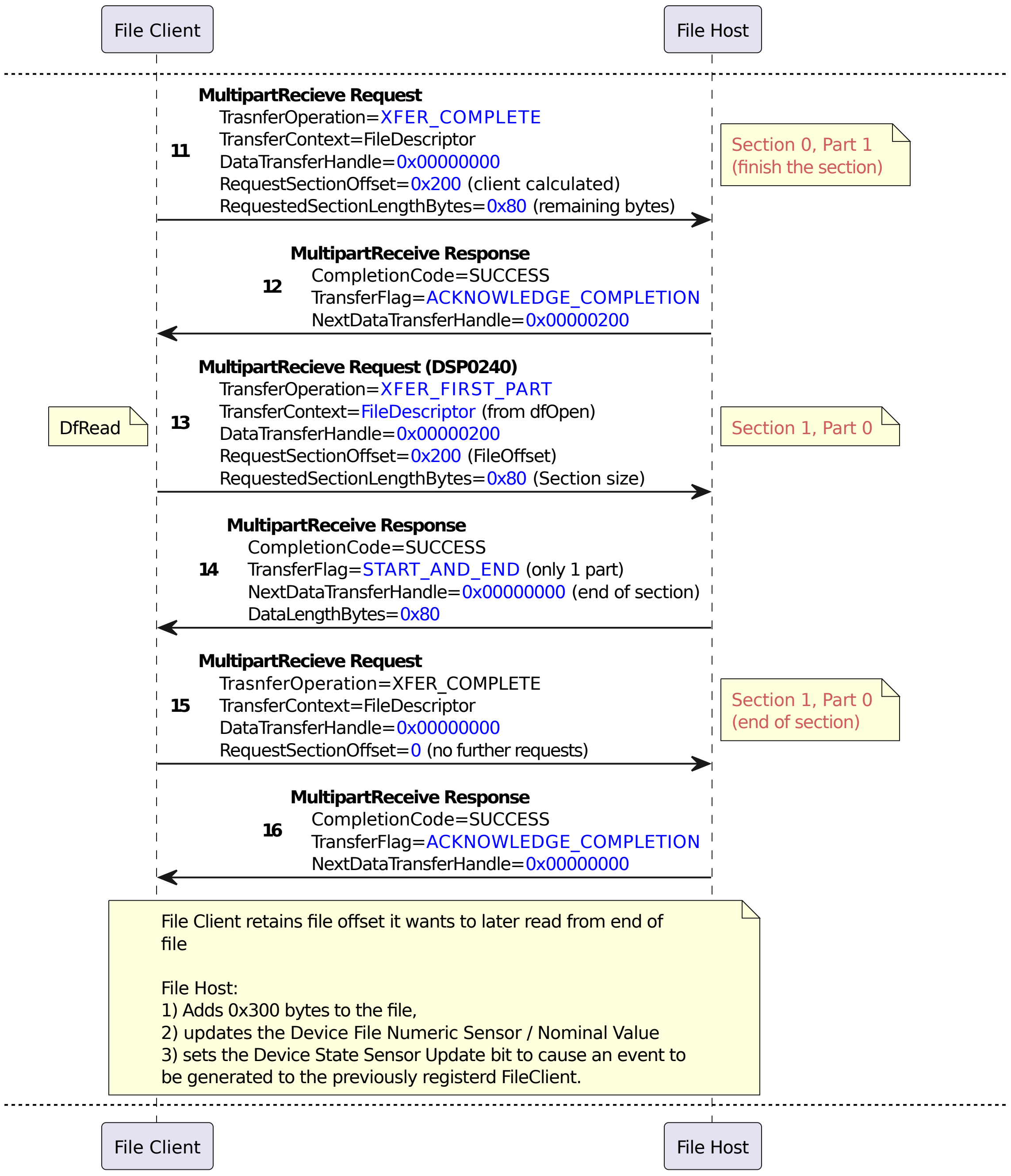

File Client
File Host
11
MultipartRecieve Request
TrasnferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x00000000
RequestSectionOffset=0x200 (client calculated)
RequestedSectionLengthBytes=0x80 (remaining bytes)
Section 0, Part 1 (finish the section)
12
MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=ACKNOWLEDGE_COMPLETION
NextDataTransferHandle=0x00000200
DfRead
13
MultipartRecieve Request (DSP0240)
TransferOperation=XFER_FIRST_PART
TransferContext=FileDescriptor (from dfOpen)
DataTransferHandle=0x00000200
RequestSectionOffset=0x200 (FileOffset)
RequestedSectionLengthBytes=0x80 (Section size)
Section 1, Part 0
14
MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=START_AND_END (only 1 part)
NextDataTransferHandle=0x00000000 (end of section)
DataLengthBytes=0x80
15
MultipartRecieve Request
TrasnferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x00000000
RequestSectionOffset=0 (no further requests)
Section 1, Part 0 (end of section)
16
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 registerd FileClient.
File Client
File Host

Figure 6 — Regular file read example - Page 2

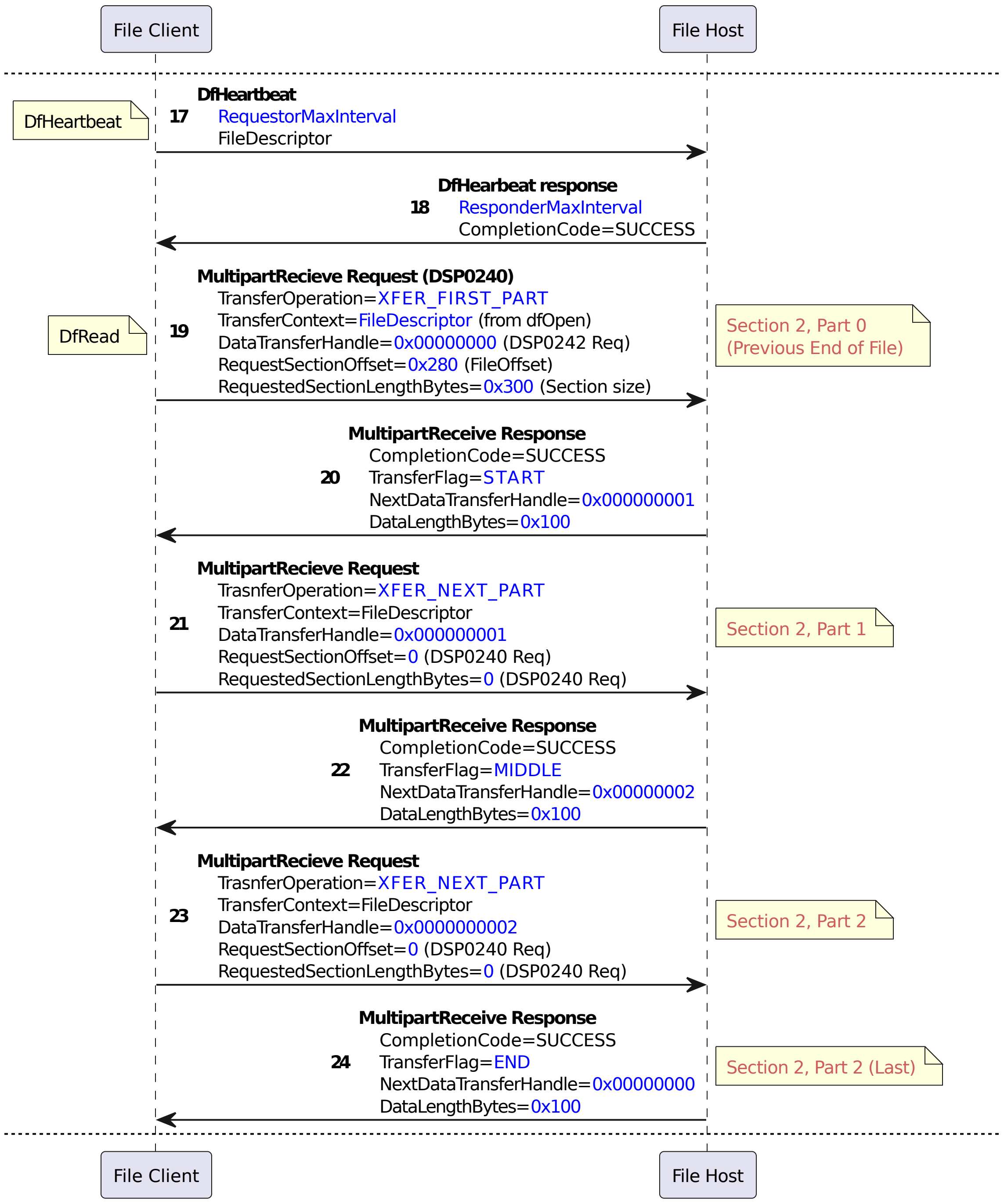

File Client
File Host
DfHeartbeat
DfHeartbeat
17 RequestorMaxInterval
FileDescriptor
DfHearbeat response
18 ResponderMaxInterval
CompletionCode=SUCCESS
MultipartRecieve Request (DSP0240)
TransferOperation=XFER_FIRST_PART
DfRead
19
TransferContext=FileDescriptor (from dfOpen)
DataTransferHandle=0x00000000 (DSP0242 Req)
RequestSectionOffset=0x280 (FileOffset)
RequestedSectionLengthBytes=0x300 (Section size)
Section 2, Part 0
(Previous End of File)
MultipartReceive Response
CompletionCode=SUCCESS
20 TransferFlag=START
NextDataTransferHandle=0x000000001
DataLengthBytes=0x100
MultipartRecieve Request
TrasnferOperation=XFER_NEXT_PART
21
TransferContext=FileDescriptor
DataTransferHandle=0x000000001
RequestSectionOffset=0 (DSP0240 Req)
RequestedSectionLengthBytes=0 (DSP0240 Req)
Section 2, Part 1
MultipartReceive Response
CompletionCode=SUCCESS
22 TransferFlag=MIDDLE
NextDataTransferHandle=0x00000002
DataLengthBytes=0x100
MultipartRecieve Request
TrasnferOperation=XFER_NEXT_PART
23
TransferContext=FileDescriptor
DataTransferHandle=0x0000000002
RequestSectionOffset=0 (DSP0240 Req)
RequestedSectionLengthBytes=0 (DSP0240 Req)
Section 2, Part 2
MultipartReceive Response
CompletionCode=SUCCESS
24 TransferFlag=END
NextDataTransferHandle=0x00000000
DataLengthBytes=0x100
Section 2, Part 2 (Last)
File Client
File Host

Figure 7 — Regular file read example - Page 3

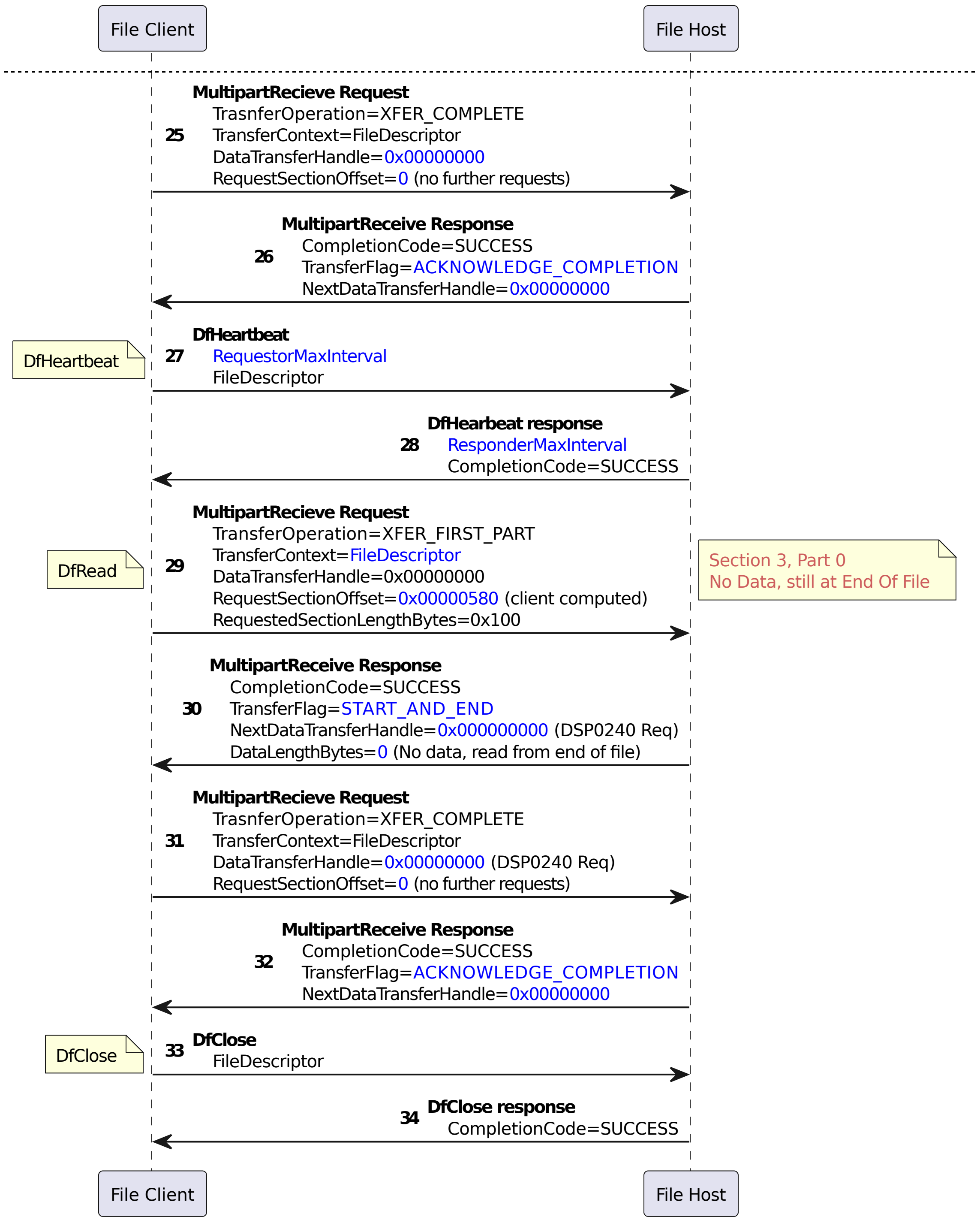

File Client
File Host
MultipartRecieve Request
25
TrasnferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x00000000
RequestSectionOffset=0 (no further requests)
MultipartReceive Response
26
CompletionCode=SUCCESS
TransferFlag=ACKNOWLEDGE_COMPLETION
NextDataTransferHandle=0x00000000
DfHeartbeat
DfHeartbeat
27
RequestorMaxInterval
FileDescriptor
DfHearbeat response
28
ResponderMaxInterval
CompletionCode=SUCCESS
MultipartRecieve Request
DfRead
29
TransferOperation=XFER_FIRST_PART
TransferContext=FileDescriptor
DataTransferHandle=0x00000000
RequestSectionOffset=0x00000580 (client computed)
RequestedSectionLengthBytes=0x100
Section 3, Part 0
No Data, still at End Of File
MultipartReceive Response
30
CompletionCode=SUCCESS
TransferFlag=START_AND_END
NextDataTransferHandle=0x000000000 (DSP0240 Req)
DataLengthBytes=0 (No data, read from end of file)
MultipartRecieve Request
31
TrasnferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x00000000 (DSP0240 Req)
RequestSectionOffset=0 (no further requests)
MultipartReceive Response
32
CompletionCode=SUCCESS
TransferFlag=ACKNOWLEDGE_COMPLETION
NextDataTransferHandle=0x00000000
DfClose
33
DfClose
FileDescriptor
DfClose response
34
CompletionCode=SUCCESS
File Client
File Host

Figure 8 — Regular file read example - Page 4

10.3 Polled serial log read

Figures 9, 10, and 11 show an example of a serial (FIFO) log read

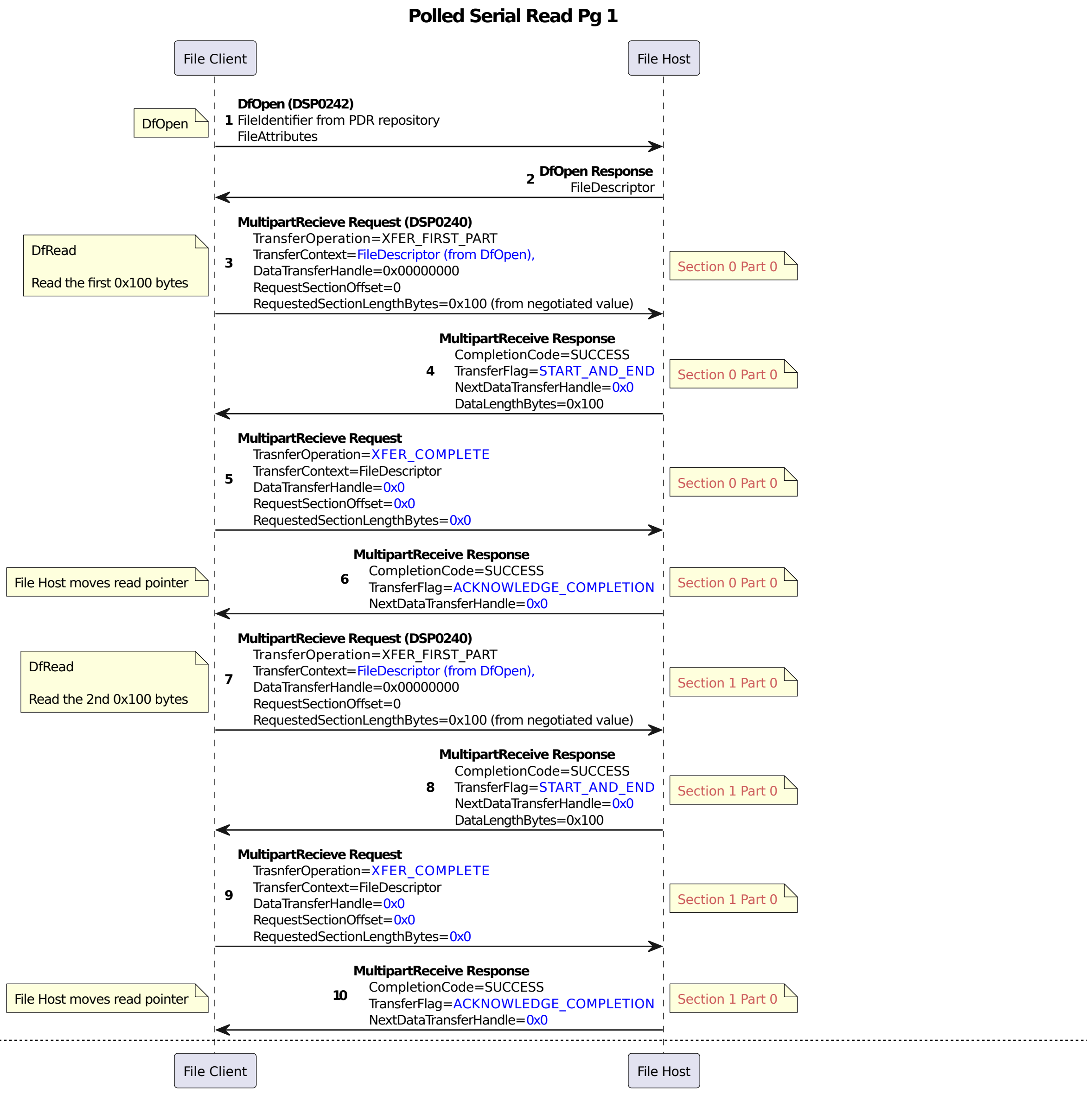

Polled Serial Read Pg 1
File Client
File Host
DfOpen
1 DfOpen (DSP0242)
FileIdentifier from PDR repository
FileAttributes
2 DfOpen Response
FileDescriptor
DfRead
Read the first 0x100 bytes
3 MultipartRecieve Request (DSP0240)
TransferOperation=XFER_FIRST_PART
TransferContext=FileDescriptor (from DfOpen),
DataTransferHandle=0x00000000
RequestSectionOffset=0
RequestedSectionLengthBytes=0x100 (from negotiated value)
Section 0 Part 0
4 MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=START_AND_END
NextDataTransferHandle=0x0
DataLengthBytes=0x100
Section 0 Part 0
5 MultipartRecieve Request
TrasnferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x0
RequestSectionOffset=0x0
RequestedSectionLengthBytes=0x0
Section 0 Part 0
File Host moves read pointer
6 MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=ACKNOWLEDGE_COMPLETION
NextDataTransferHandle=0x0
Section 0 Part 0
DfRead
Read the 2nd 0x100 bytes
7 MultipartRecieve Request (DSP0240)
TransferOperation=XFER_FIRST_PART
TransferContext=FileDescriptor (from DfOpen),
DataTransferHandle=0x00000000
RequestSectionOffset=0
RequestedSectionLengthBytes=0x100 (from negotiated value)
Section 1 Part 0
8 MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=START_AND_END
NextDataTransferHandle=0x0
DataLengthBytes=0x100
Section 1 Part 0
9 MultipartRecieve Request
TrasnferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x0
RequestSectionOffset=0x0
RequestedSectionLengthBytes=0x0
Section 1 Part 0
File Host moves read pointer
10 MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=ACKNOWLEDGE_COMPLETION
NextDataTransferHandle=0x0
Section 1 Part 0
File Client
File Host

Figure 9 — Polled serial read example - Page 1

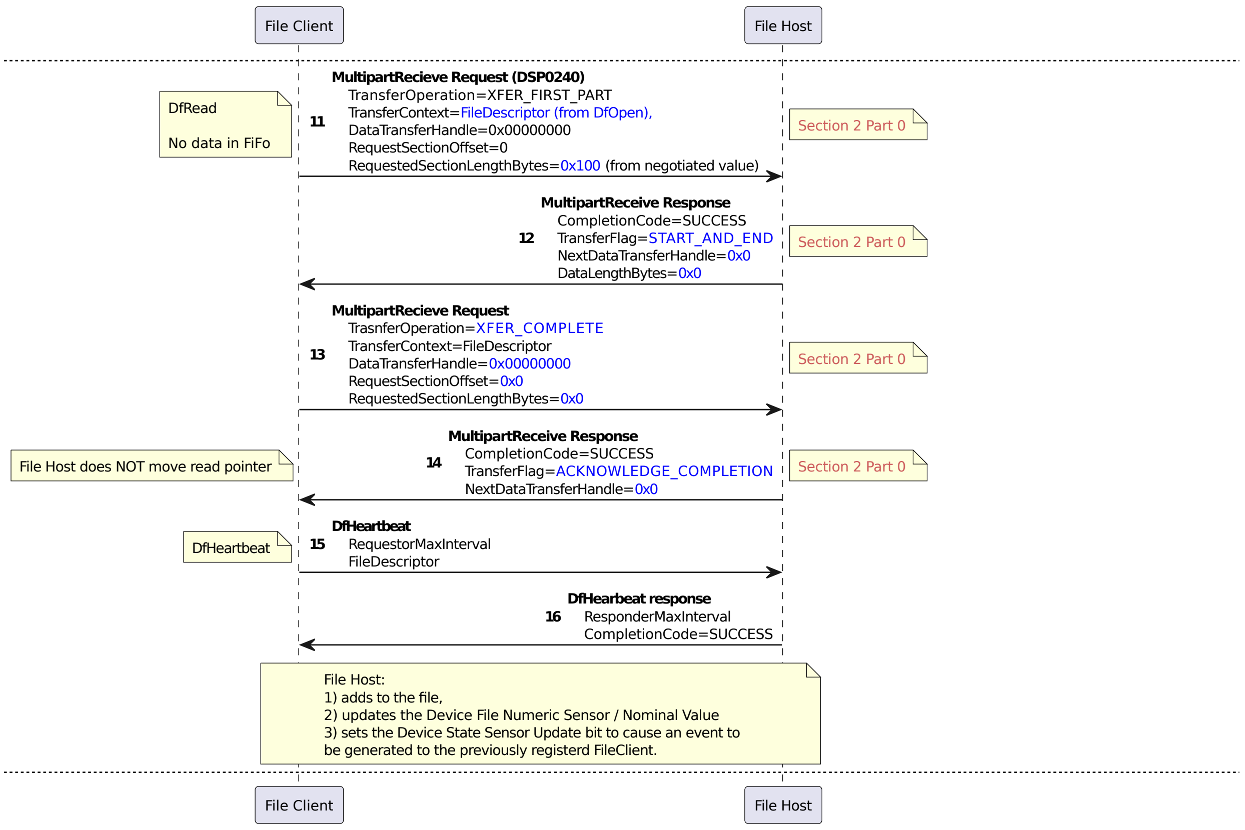
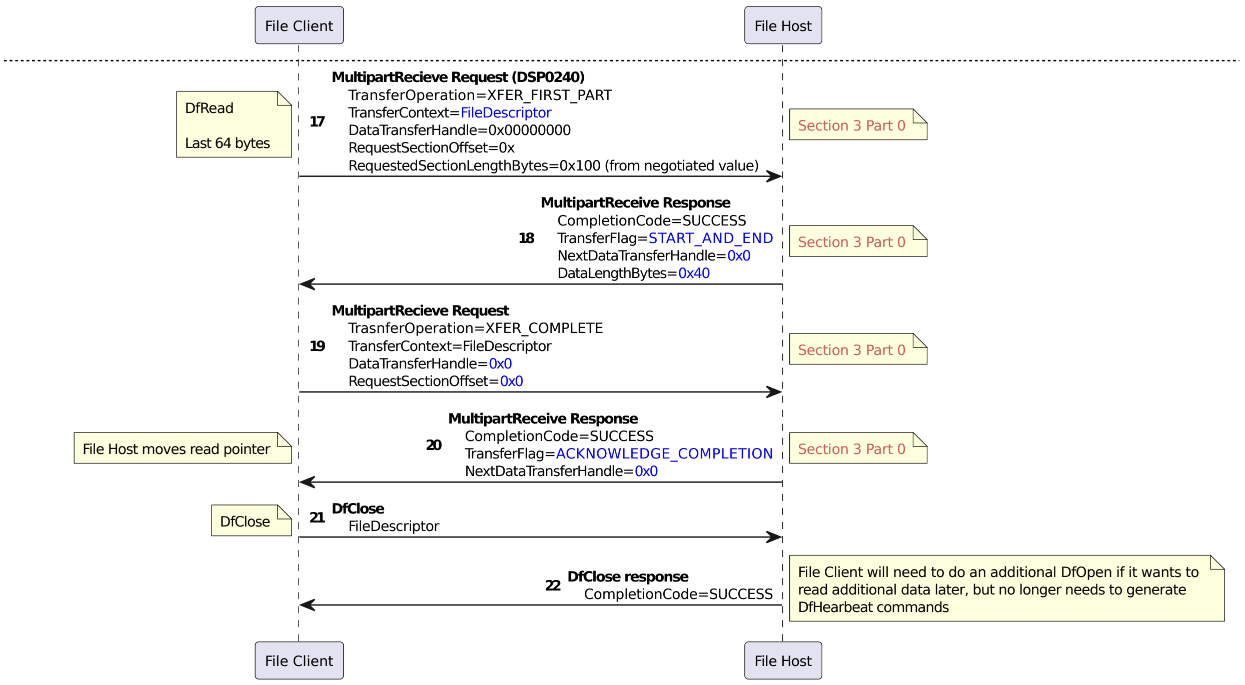

File Client
File Host
MultipartRecieve Request (DSP0240)
TransferOperation=XFER_FIRST_PART
TransferContext=FileDescriptor (from DfOpen),
DataTransferHandle=0x00000000
RequestSectionOffset=0
RequestedSectionLengthBytes=0x100 (from negotiated value)
11
DfRead
No data in FiFo
Section 2 Part 0
MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=START_AND_END
NextDataTransferHandle=0x0
DataLengthBytes=0x0
12
Section 2 Part 0
MultipartRecieve Request
TrasnferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x00000000
RequestSectionOffset=0x0
RequestedSectionLengthBytes=0x0
13
Section 2 Part 0
MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=ACKNOWLEDGE_COMPLETION
NextDataTransferHandle=0x0
14
File Host does NOT move read pointer
Section 2 Part 0
DfHeartbeat
RequestorMaxInterval
FileDescriptor
15
DfHeartbeat
DfHearbeat response
ResponderMaxInterval
CompletionCode=SUCCESS
16
File Host:
1) adds 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 registerd FileClient.
File Client
File Host

Figure 10 — Polled serial read example - Page 2


File Client
File Host
DfRead

Last 64 bytes
17 MultipartRecieve Request (DSP0240)
TransferOperation=XFER_FIRST_PART
TransferContext=FileDescriptor
DataTransferHandle=0x00000000
RequestSectionOffset=0x
RequestedSectionLengthBytes=0x100 (from negotiated value)
Section 3 Part 0
18 MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=START_AND_END
NextDataTransferHandle=0x0
DataLengthBytes=0x40
Section 3 Part 0
19 MultipartRecieve Request
TrasnferOperation=XFER_COMPLETE
TransferContext=FileDescriptor
DataTransferHandle=0x0
RequestSectionOffset=0x0
Section 3 Part 0
File Host moves read pointer
20 MultipartReceive Response
CompletionCode=SUCCESS
TransferFlag=ACKNOWLEDGE_COMPLETION
NextDataTransferHandle=0x0
Section 3 Part 0
DfClose
21 DfClose
FileDescriptor
22 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 DfHearbeat commands
File Client
File Host

Figure 11 — Polled serial read example - Page 3

10.4 Async serial log read

Figures 12 and 13 shows an example of a Async serial (FIFO) log read

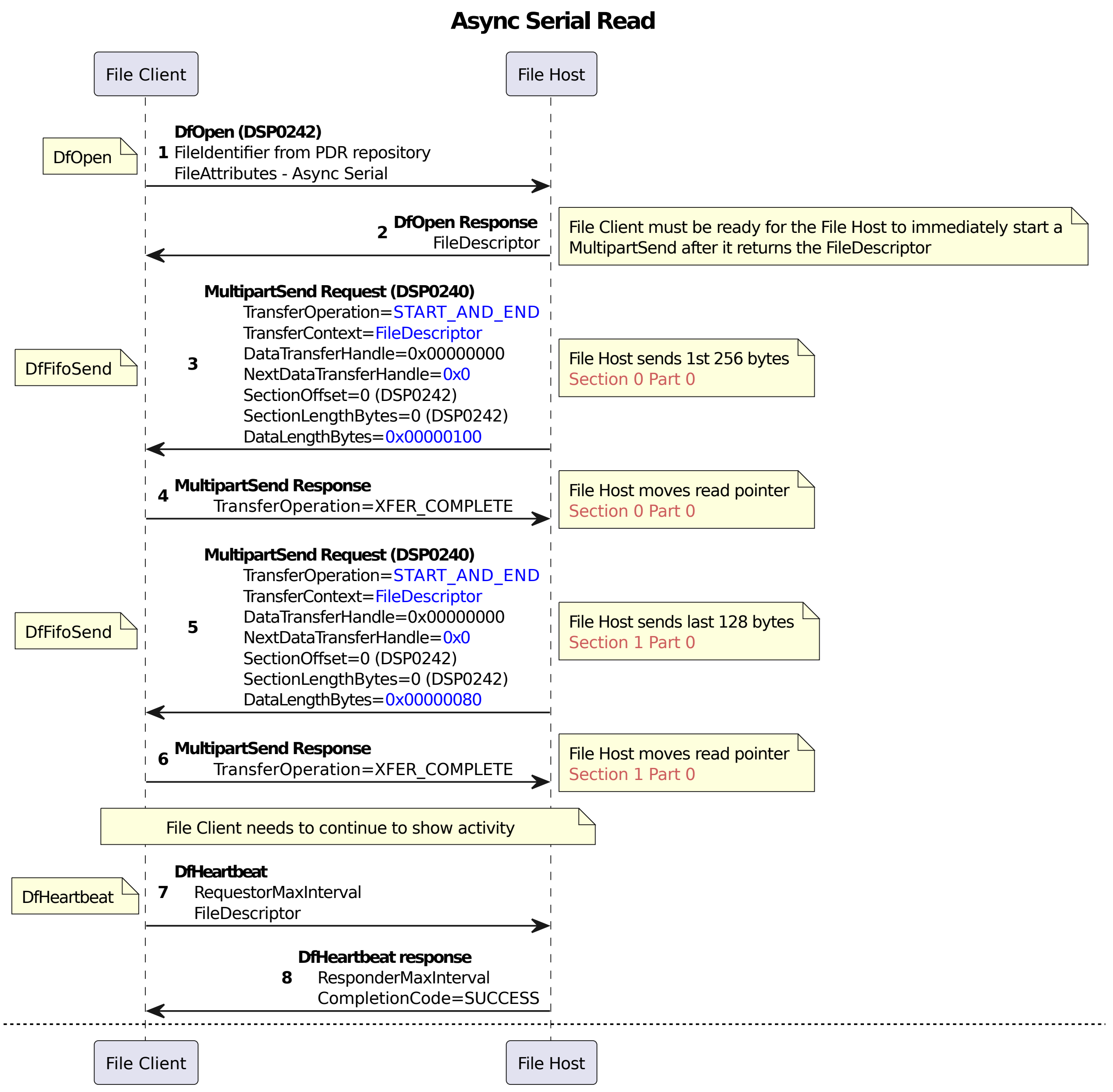

Async Serial Read
File Client
File Host
DfOpen
1 DfOpen (DSP0242)
FileIdentifier from PDR repository
FileAttributes - Async Serial
2 DfOpen Response
FileDescriptor
File Client must be ready for the File Host to immediately start a MultipartSend after it returns the FileDescriptor
DfFifoSend
3 MultipartSend 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
DfFifoSend
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
File Client needs to continue to show activity
DfHeartbeat
7 DfHeartbeat
RequestorMaxInterval
FileDescriptor
8 DfHeartbeat response
ResponderMaxInterval
CompletionCode=SUCCESS
File Client
File Host

Figure 12 — Async serial read example - Page 1

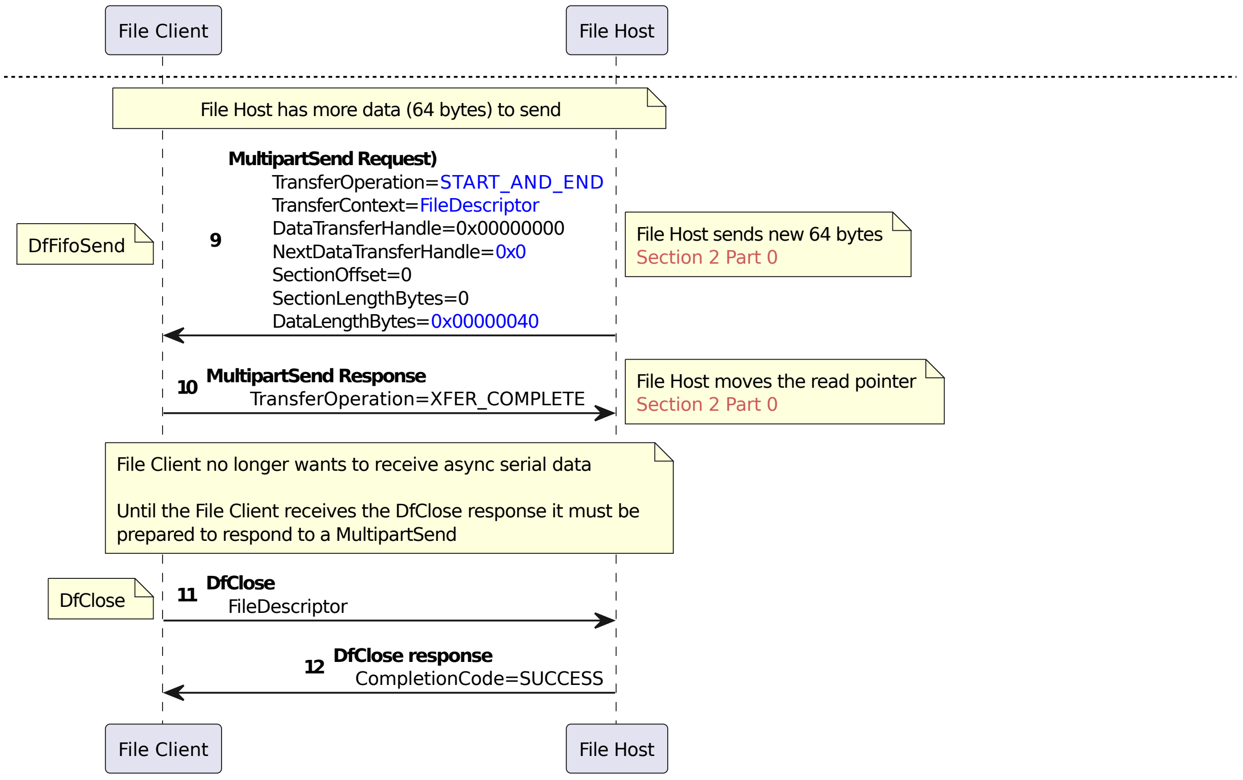

File Client
File Host
File Host has more data (64 bytes) to send
9
MultipartSend Request)
TransferOperation=START_AND_END
TransferContext=FileDescriptor
DataTransferHandle=0x00000000
NextDataTransferHandle=0x0
SectionOffset=0
SectionLengthBytes=0
DataLengthBytes=0x00000040
DfFifoSend
File Host sends new 64 bytes
Section 2 Part 0
10
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 DfClose response it must be prepared to respond to a MultipartSend
DfClose
11
DfClose
FileDescriptor
12
DfClose response
CompletionCode=SUCCESS
File Client
File Host

Figure 13 — Async serial read example - Page 2