Document Identifier: DSP0267

Date: 2018-03-28

Version: 1.0.1

Platform Level Data Model (PLDM) for Firmware Update Specification

Supersedes: 1.0.0

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patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a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 the DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit http://www.dmtf.org/about/policies/disclosures.php.

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

CONTENTS

Figures

Tables

Foreword

The *Update Specification* (DSP0267) was prepared by the Platform Management Components Intercommunications (PMCI Working Group) of the DMTF.

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.

Acknowledgments

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

Editor:

- Patrick Caporale – Lenovo

Contributors:

- Scott Dunham – Lenovo
- Kaijie Guo – Lenovo
- Yuval Itkin – Mellanox Technologies
- Shai Lazmi – QLogic Corporation
- Eliel Louzoun – Intel Corporation
- Rob Mapes – QLogic Corporation
- Edward Newman - Hewlett Packard Enterprise
- Patrick Schoeller – Hewlett Packard Enterprise
- Hemal Shah – Broadcom Limited
- James Smart – Broadcom Limited
- Bob Stevens – Dell

Introduction

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

Document conventions

Typographical conventions

The following typographical conventions are used in this document:

- Document titles are marked in *italics*.

Platform Level Data Model (PLDM) for Firmware Update Specification

1 Scope

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

This specification is not a system-level requirements document. The mandatory requirements stated in this specification apply when a particular capability is implemented through PLDM messaging in a manner that is conformant with this specification. This specification does not specify whether a given system is required to implement that capability. For example, this specification does not specify whether a given system shall support firmware updates over PLDM. However, if a system does support firmware updates over PLDM or other functions described in this specification, the specification defines the requirements to access and use those functions over PLDM.

Portions of this specification rely on information and definitions from other specifications, which are identified in clause 2. Two of these references are particularly relevant:

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

2 Normative references

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

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

DMTF DSP0236, *MCTP Base Specification 1.2.0*,
http://dmtf.org/sites/default/files/standards/documents/DSP0236_1.2.0.pdf

DMTF DSP0240, *Platform Level Data Model (PLDM) Base Specification 1.0*,
http://dmtf.org/sites/default/files/standards/documents/DSP0240_1.0.0.pdf

DMTF DSP0241, *Platform Level Data Model (PLDM) Over MCTP Binding Specification 1.0*,
http://dmtf.org/sites/default/files/standards/documents/DSP0241_1.0.0.pdf

DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes Specification 1.2.0*,
http://dmtf.org/sites/default/files/standards/documents/DSP0245_1.2.0.pdf

DMTF DSP0248, *Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification 1.1.0*, http://dmtf.org/sites/default/files/standards/documents/DSP0248_1.1.0.pdf

DMTF DSP0249, *Platform Level Data Model (PLDM) State Sets Specification 1.0,* http://dmtf.org/sites/default/files/standards/documents/DSP0249_1.0.0.pdf

DMTF DSP0257, *Platform Level Data Model (PLDM) FRU Data Specification 1.0,* http://dmtf.org/sites/default/files/standards/documents/DSP0257_1.0.0.pdf

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

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

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

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

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

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards,* http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

3 Terms and definitions

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

3.1

activation

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

3.2

auto-apply

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

3.3

automatic activation

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

3.4

AC power cycle

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

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

3.5

AC power cycle activation

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

3.6

code image

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

3.7

component classification

the general type of component

Values for this field are aligned with the Value Map from CIM_SoftwareIdentify.Classifications. Refer to Table 19 for values.

3.8

component comparison stamp

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

3.9

component identifier

a vendor defined value that distinguishes between firmware components that may have identical classifications but require different component images

3.10

component image

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

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

3.11

component image set

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

3.12

device identifier record

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

3.13

DC power cycle

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

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

3.14

DC power cycle activation

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

3.15

firmware

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

3.16

firmware device

FD

a PLDM endpoint (terminus) that contains one or more processor elements that execute firmware

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

3.17

firmware component

a logical entity representing a functional portion of a firmware device

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

3.18

firmware package header

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

3.19

firmware update baseline transfer size

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

3.20

firmware update package

a firmware package header describing the contents concatenated with one or more component images for one or more firmware devices

3.21

medium-specific reset

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

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

3.22

pending firmware component

a newly transferred component image has been fully delivered to the firmware device, but is not the actively running code image

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

3.23

self-contained activation

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

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

3.24

software bundle

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

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

3.25

system reboot

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

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

3.26

update agent

UA

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

Typically this agent is contained within a management controller

4 Symbols and abbreviated terms

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.

4.1

FD

Firmware Device

4.2

UA

Update Agent

5 Conventions

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

5.1 Reserved and unassigned values

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

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

5.2 Byte ordering

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

6 PLDM for firmware update overview

This specification describes the operation and format of request messages (also referred to as commands) and response messages for updating firmware components of a firmware device (FD) contained within a platform management subsystem. These messages are designed to be delivered using PLDM messaging. This specification also permits a subset of commands to be implemented by a firmware device that only supports the reporting of existing firmware component details, without the ability to perform a firmware update.

Traditionally, device firmware has been updated by a combination of update tools and binary files provided by individual device manufacturers. Those update tools normally operate inside a host operating system (e.g., Linux/Windows/DOS), whereby each device may have their own method provided by the device manufacturers to update the firmware into flash chips on the device board. This specification identifies a common method to use PLDM messaging for transferring one or more component images to an FD within the PLDM subsystem and thereby avoiding the use of host operating system based tools and utilities.

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

- Component Image Transfer
  - Component image transfer mechanism does not require FD specific logic in the UA
  - For an individual firmware device, a firmware update package may contain
    - A single combined component image (component classification of Software Bundle)
    - A single component image for a single firmware component
    - Multiple component images for multiple firmware components that are applicable to the same firmware device
  - Transfer of a component image is requested through an offset-based method as directed by the FD

- Firmware Update Package to Firmware Device association
  - A mechanism to determine which type of FD a firmware update package is targeted
  - A mechanism to distinguish between firmware update packages applicable to different instantiations of the same FD (e.g., planar vs. adapter)
  - A mechanism to identify the component image that is to be transferred based on device identifier records. A device identifier record may be based on PCI IDs, IANA ID, UUID, or a vendor specific ID.
- Activation Requirements Gathering
  - A mechanism to learn the activation requirements of the FD firmware components
  - This will allow more timely and coordinated activation of all firmware components in the system
  - Activation requirements for self-activation capable firmware devices shall specify recovery times

6.1 Firmware update concepts

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

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

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

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

- Classification: identifies the function type of the component image, such as UEFI driver, port controller firmware, update SW, diagnostic code, firmware bundle, etc.
- Identifier: A unique value (per vendor) that distinguishes between component images that may have identical classifications but contain different code images.
- Component Comparison Stamp: An optional vendor-assigned value that can be used to compare levels between the firmware component within the FD and the component image within the firmware update package. For example, an FD vendor might use a value for this field in the format of MajorMinorRevisionPatch where each subfield has a range of 0x00 to 0xFF. The component comparison stamp if implemented shall contain a value that can be compared to another component comparison stamp using an unsigned integer compare. Therefore when comparing component comparison stamps the lower value is down-level compared to the other when performing an unsigned integer comparison between the two.
- Version: Contains a string describing the component image version. The version string for the component image is provided by the FD vendor.

6.2 Update Agent

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

6.3 PLDM firmware update packaging

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

To assist in performing an update over PLDM, the firmware update package shall contain a firmware package header describing the contents of the firmware update package. The header shall include (refer to clause 7 for details of the header structure):

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

6.4 Update flow overview

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

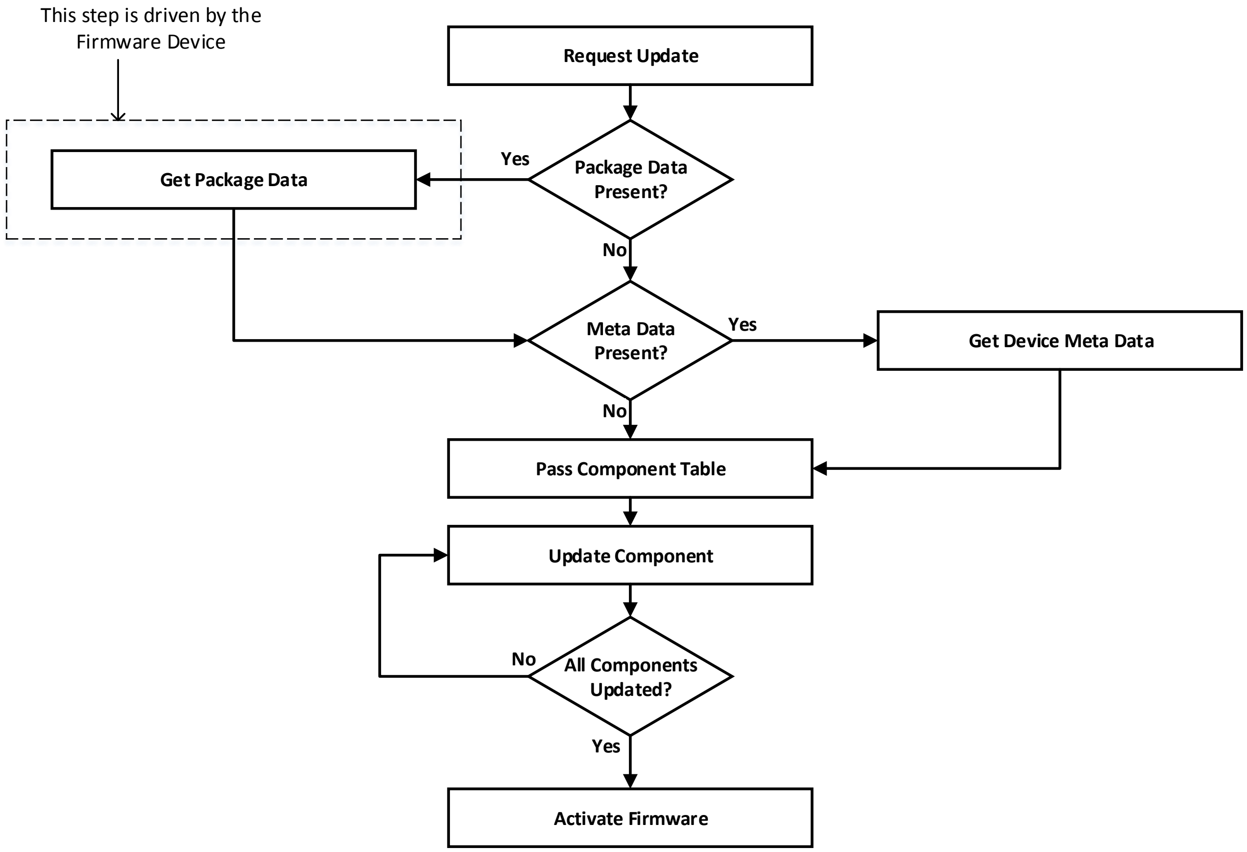

Figure 1 – High level firmware update flow

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

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

2) If the firmware update package contains optional package data for the firmware device, then the UA shall transfer the package data to the FD prior to transferring component images. Refer to clause 7 for more details about the optional package data.

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

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

5) The UA processes each of the applicable component images in the firmware update package one by one in the same sequence as is described in the firmware package header. The detailed steps of updating a component are described in clause 6.5.

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

Note that for components that do not support self-contained activation, the ActivateFirmware command instructs the FD to perform FD-specific actions required to set the remaining updated firmware components into a 'pending activation' state. The newly transferred component images will then become the actively running code images upon external activation (such as a medium specific reset or a host reboot). Non-self-contained activation can be scheduled for a later time via a procedure that is not defined within this specification.

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

2) If the UA is no longer able to communicate with the FD in order to cancel update mode, the FD itself shall provide an internal timer to exit from update mode if no commands are received. Refer to FD_T1 in clause 6.10 of this document. If the FD had begun the apply or activate step, then it shall finish that operation before exiting from update mode, otherwise the FD should attempt to discard the component image and exit from update mode.

6.5 Detailed steps of updating a firmware component

The steps below define transactions required to update one firmware component. If there is any error or timeout during the transfer of a component image, the timing specifications defined within DSP0240 shall be followed for command response timeouts and retries. In addition, specific PLDM Firmware Update timing specifications are defined in clause 6.10 and shall be followed.

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

2) The FD proceeds to request the component image, by sending one or more RequestFirmwareData commands to the UA. The request command specifies a component image portion to be transferred via the offset and length fields in the RequestFirmwareData command. The UA will validate the request, and if within the permitted range of the component image defined by the firmware package header and additional padding, generate a successful response containing the component image portion requested by the FD. Refer to Table 21 for details on the permitted range for the request.

The size of the component image portion requested shall:

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

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

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

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

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

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

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

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

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

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

1) The UA may send the CancelUpdateComponent command to cancel the update of the current component image being transferred. If the FD has currently requested a portion of component image data via the RequestFirmwareData command, the UA should first respond to any outstanding RequestFirmwareData commands received before sending its request to

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

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

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

Other timeouts or retries may occur and the timing specification defined within clause 6.10 shall be followed.

Figure 2 shows the flow for updating a single firmware component.

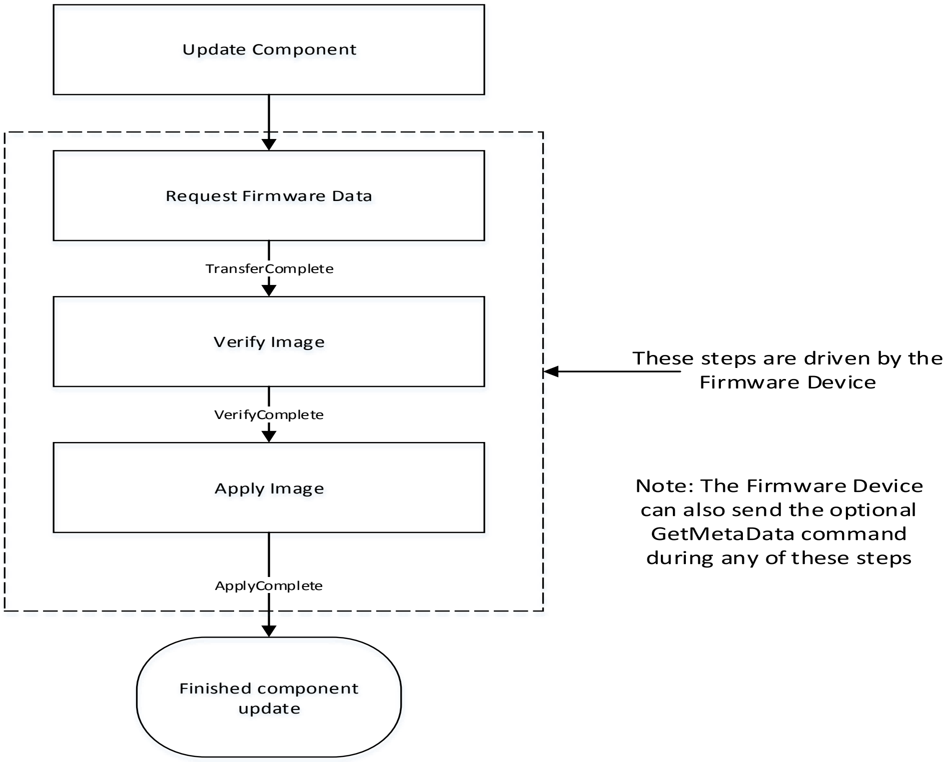
Update Component
Request Firmware Data
TransferComplete
Verify Image
VerifyComplete
Apply Image
ApplyComplete
Finished component update
These steps are driven by the Firmware Device
Note: The Firmware Device can also send the optional GetMetaData command during any of these steps

Figure 2 – Firmware component update flow

6.6 Firmware update baseline transfer size

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

6.7 Firmware component authentication

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

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

6.8 Type code

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

6.9 Error completion codes

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

Table 1 – PLDM firmware update completion codes

Value	Name	Returned By	Description
Various	PLDM_BASE_CODES	FD & UA	Refer to DSP0240 for a full list of PLDM Base Code Completion values that are supported.
0x80	NOT_IN_UPDATE_MODE	FD	Received PLDM firmware update command when the FD is not in update mode.
0x81	ALREADY_IN_UPDATE_MODE	FD	Firmware device receives RequestUpdate when it’s already in update mode.
0x82	DATA_OUT_OF_RANGE	UA	The requested component image portion has an initial offset which is not contained within the image data, or the offset plus the length requested exceeds the range permitted by the UA.
0x83	INVALID_TRANSFER_LENGTH	UA	The length of the requested component image portion exceeds the MaximumTransferSize negotiated in the RequestUpdate command, or is less than the firmware update baseline transfer size.
0x84	INVALID_STATE_FOR_COMMAND	FD	The FD is not in a state to expect this command.

Value	Name	Returned By	Description
0x85	INCOMPLETE_UPDATE	FD	One or more component transfers failed to complete.
0x86	BUSY_IN_BACKGROUND	FD	The FD is performing critical background task and cannot execute the command.
0x87	CANCEL_PENDING	UA	Sent by the UA when it receives a RequestFirmwareData command after sending a CancelUpdate or CancelUpdateComponent command.
0x88	COMMAND_NOT_EXPECTED	UA	Sent by the UA when it receives a command from the FD out of sequence from when it is expected.
0x89	RETRY_REQUEST_FW_DATA	UA	The Update Agent has requested a retry of the RequestFirmwareData command as it needs more time to retrieve the section of firmware to transfer.
0x8A	UNABLE_TO_INITIATE_UPDATE	FD	The Firmware Device is not able to enter into update mode to begin a transfer.
0x8B	ACTIVATION_NOT_REQUIRED	FD	The firmware device already has enabled the firmware components to become the active running image on the next external activation, or the firmware components are already activated.
0x8C	SELF_CONTAINED_ACTIVATION_NOT_PERMITTED	FD	The firmware device does not permit Self-Contained activation and returns this code when the UA requests a self-contained activation.
0x8D	NO_DEVICE_METADATA	FD	The Firmware Device has no meta data that must be retrieved by the UA prior to the start of the component image transfers.
0x8E	RETRY_REQUEST_UPDATE	FD	The Firmware Device has requested a retry of the RequestUpdate command as it needs more time to prepare for a firmware update.
0x8F	NO_PACKAGE_DATA	UA	The Update Agent has no package data available for the firmware device
0x90	INVALID_DATA_TRANSFER_HANDLE	FD & UA	The data transfer handle requested was invalid
0x91	INVALID_TRANSFER_OPERATION_FLAG	FD & UA	The transfer operation flag used in the request was invalid

6.10 Timing specification

Table 2 below defines timing values that are specific to this document. The table below defines the timing parameters defined for the PLDM Firmware Update Specification. In addition, all timing parameters listed in DSP0240 for command timeouts and number of retries shall also be followed. Figure 3 provides a visual representation example of how the minimum and maximum timing parameters should be implemented.

Table 2 – Timing specification

Timing specification	Applicable to UA or FD	Symbol	Min	Max	Description
PLDM Base Timing	UA & FD	PNx PTx			Refer to DSP0240 for the details on these timing values that are applicable to PLDM message timeouts where a response is not received by the UA or FD after sending a request.
Number of request retries when a response is received that requires a retry	UA & FD	UAFD_ T1	2		Total of three tries, minimum: the original try plus two retries.
Update mode idle timeout	FD	FD_T1	60 seconds	120 seconds	Amount of time before the FD shall exit from update mode if no command is received from the Update Agent when it's expected, during the firmware update process. For example, the FD shall wait a minimum of 60 seconds for the UA to send a PassComponentTable or UpdateComponent command.
Retry request for firmware data	FD	FD_T2	1 second	5 seconds	Amount of time for the FD to wait before resending a RequestFirmwareData command after receiving a RETRY_REQUEST_FW_DATA code from the UA.
Retry interval to send next cancel command	UA	UA_T1	500 milliseconds	5 seconds	Amount of time to wait before the UA sends an additional CancelUpdate or CancelUpdateComponent command.
Request firmware data idle timeout	UA	UA_T2	60 seconds	90 seconds	Amount of time for the Update Agent to cancel the component update if no command is received from the FD when it's expected, during the component image transfer stage. For example, the UA shall wait a minimum of 60 seconds for the FD to send another RequestFirmwareData command.
State change timeout	UA	UA_T3	180 seconds	-	Amount of time for the Update Agent to wait before canceling the component update if the ProgressPercent value in the GetStatus command remains unchanged.
Retry request for update	UA	UA_T4	1 second	5 seconds	Amount of time for the UA to wait before resending a RequestUpdate command after receiving a RETRY_REQUEST_UPDATE code from the FD.

Timing specification	Applicable to UA or FD	Symbol	Min	Max	Description
Get Package Data timeout	UA	UA_T5	1 second	5 seconds	Amount of time for the UA to wait to receive the GetPackageData command if the FD indicated that it would send that command in the response to RequestUpdate. The UA shall send CancelUpdate if this timer expires.

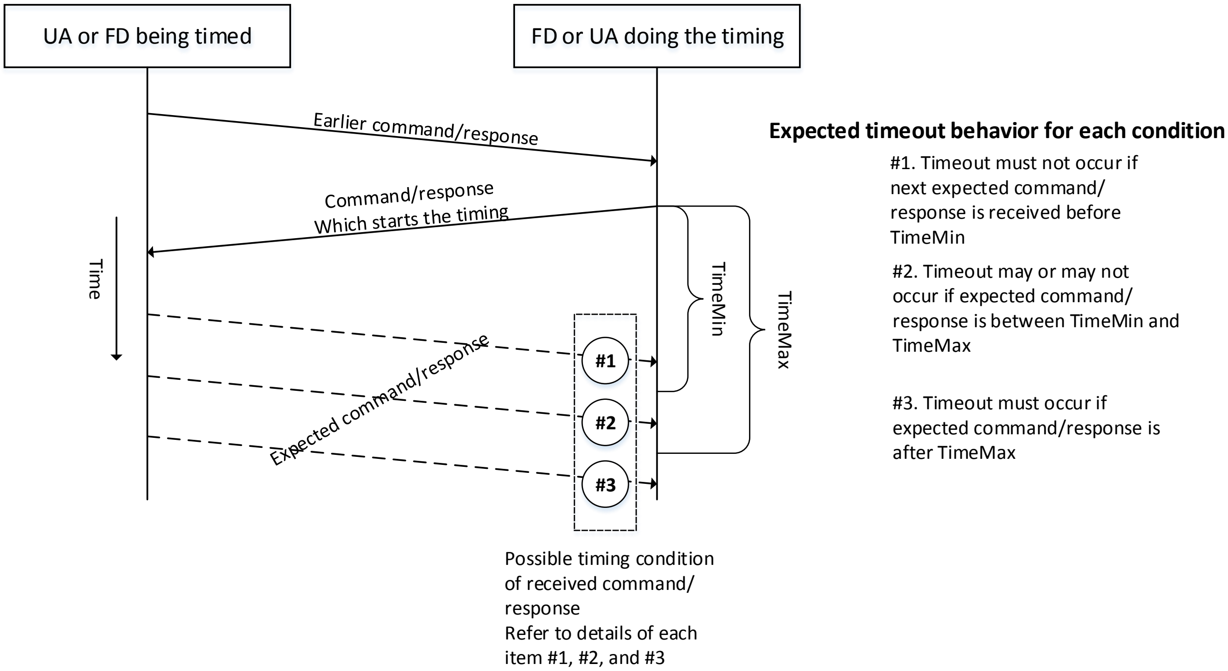
UA or FD being timed
FD or UA doing the timing
Earlier command/response
Command/response
Which starts the timing
Time
Expected command/response
#1
#2
#3
TimeMin
TimeMax
Expected timeout behavior for each condition
#1. Timeout must not occur if next expected command/ response is received before TimeMin
#2. Timeout may or may not occur if expected command/ response is between TimeMin and TimeMax
#3. Timeout must occur if expected command/response is after TimeMax
Possible timing condition of received command/ response
Refer to details of each item #1, #2, and #3

Figure 3 – Timeout behavior diagram

7 PLDM firmware update package

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

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

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

The firmware update header supports the following:

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

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

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

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

- One to N explicit component images

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

Figure 4 shows the entire firmware update package:

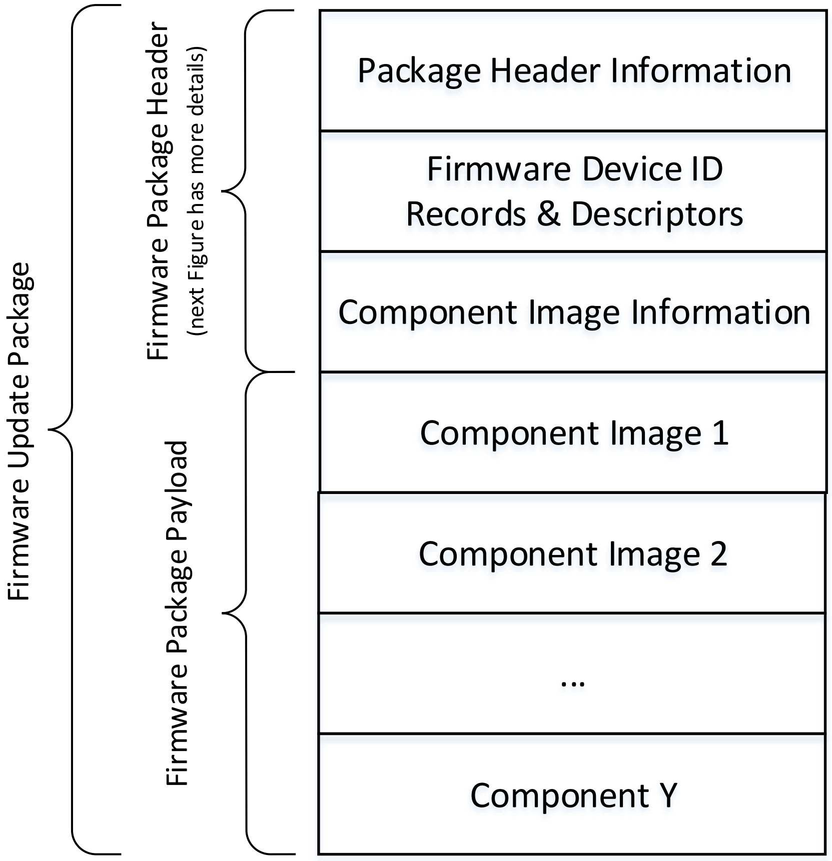
Firmware Update Package
Firmware Package Header
(next Figure has more details)
Firmware Package Payload
Package Header Information
Firmware Device ID Records & Descriptors
Component Image Information
Component Image 1
Component Image 2
...
Component Y

Figure 4 – PLDM firmware update package

Figure 5 shows the structures within the firmware package header:

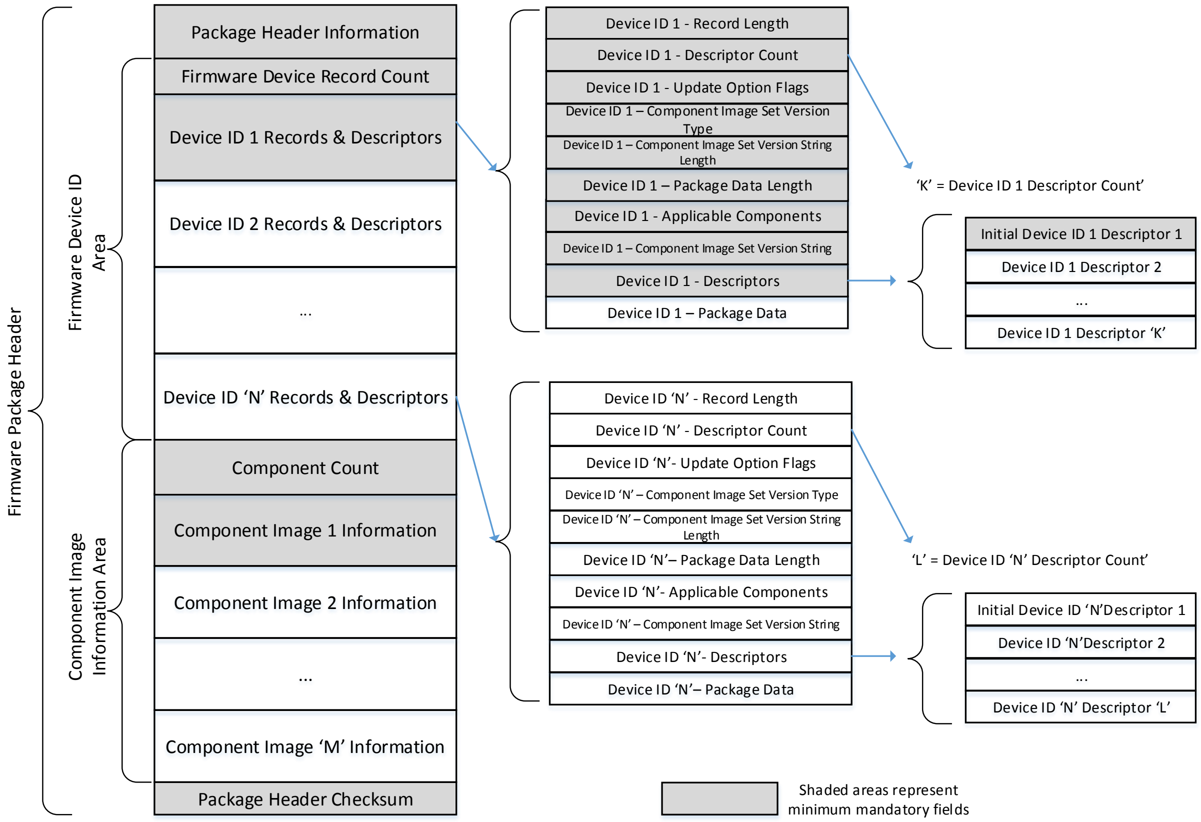
Firmware Package Header
Firmware Device ID Area
Component Image Information Area
Package Header Information
Firmware Device Record Count
Device ID 1 Records & Descriptors
Device ID 2 Records & Descriptors
...
Device ID 'N' Records & Descriptors
Component Count
Component Image 1 Information
Component Image 2 Information
...
Component Image 'M' Information
Package Header Checksum
Device ID 1 - Record Length
Device ID 1 - Descriptor Count
Device ID 1 - Update Option Flags
Device ID 1 – Component Image Set Version Type
Device ID 1 – Component Image Set Version String Length
Device ID 1 – Package Data Length
Device ID 1 - Applicable Components
Device ID 1 – Component Image Set Version String
Device ID 1 - Descriptors
Device ID 1 – Package Data
'K' = Device ID 1 Descriptor Count'
Initial Device ID 1 Descriptor 1
Device ID 1 Descriptor 2
...
Device ID 1 Descriptor 'K'
Device ID 'N' - Record Length
Device ID 'N' - Descriptor Count
Device ID 'N'- Update Option Flags
Device ID 'N' – Component Image Set Version Type
Device ID 'N' – Component Image Set Version String Length
Device ID 'N'– Package Data Length
Device ID 'N'- Applicable Components
Device ID 'N' – Component Image Set Version String
Device ID 'N'- Descriptors
Device ID 'N'– Package Data
'L' = Device ID 'N' Descriptor Count'
Initial Device ID 'N'Descriptor 1
Device ID 'N'Descriptor 2
...
Device ID 'N' Descriptor 'L'
Shaded areas represent minimum mandatory fields

Figure 5 – PLDM firmware package header structure

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

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

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

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

The firmware package payload contains the individual component images that can be transferred to the firmware devices. Prior to transferring the component images, the header shall be parsed by the UA to identify the following:

- Determine if the firmware update package is applicable for updating a specific FD by comparing device identifier records in the package header to those obtained from the FD via the QueryDeviceIdentifiers command.
- Locate the component image for each firmware component if multiple components are contained in the firmware update package. A bitmap of which packaged components are intended for which matched FDs is also contained in the header.

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

Table 3 – PLDM firmware package header

<table>
<tr><th colspan="2">Package Header Information
Byte ordering for applicable header fields is Little Endian per clause 5.2</th></tr>
<tr><th>Type</th><th>Definition</th></tr>
<tr><td>UUID</td><td><b>PackageHeaderIdentifier</b>
Mandatory label that defines this object as a valid PLDM Firmware Update Package that includes a formatted header that complies with this specification.
F018878CCB7D49439800A02F059ACA02 is the value to be used for this field that will identify the package as one that supports this PLDM Firmware Update specification.
UUID field is Big Endian. Refer to the PLDM Base Specification for field format definition.</td></tr>
<tr><td>uint8</td><td><b>PackageHeaderFormatRevision</b>
The revision number of the header structure itself. Updated when any field in the PLDM Firmware Update Header changes.
Current definition is value 0x01.
All other values are Reserved.</td></tr>
<tr><td>uint16</td><td><b>PackageHeaderSize</b>
The count of all bytes in this header structure including the fields contained within the Package Header Information, Firmware Device Identification Area, Component Image Information Area, and the Package Header Checksum sections.</td></tr>
<tr><td>timestamp 104</td><td><b>PackageReleaseDateTime</b>
The date and time in which this package was released.
Refer to the PLDM Base Specification for field format definition.</td></tr>
<tr><td>uint16</td><td><b>ComponentBitmapBitLength</b>
The number of bits that will be used to represent the bitmap in the ApplicableComponents field for a matching device. The value shall be a multiple of 8 and be large enough to contain a bit for each component in the package.</td></tr>
<tr><td>enum8</td><td><b>PackageVersionStringType</b>
The type of string used in the PackageVersionString field.
Refer to Table 20 for values.</td></tr>
<tr><td>uint8</td><td><b>PackageVersionStringLength</b>
The length, in bytes, of the PackageVersionString field.</td></tr>
<tr><td>Variable</td><td><b>PackageVersionString</b>
Package version information, up to 255 bytes.
Contains a variable type string describing the version of this firmware update package.</td></tr>
</table>

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

699 The contents of the FirmwareDeviceRecords field is described in Table 4.

700 **Table 4 – Firmware Device ID record**

Individual Firmware Device ID Record (this section is repeated for each Firmware Device ID)	
Type	**Definition**
uint16	**RecordLength** The total length in bytes for this record. The length shall include the RecordLength, DescriptorCount, DeviceUpdateOptionFlags, ComponentImageSetVersionStringType, ComponentSetVersionStringLength, FirmwareDevicePackageDataLength, ApplicableComponents, ComponentImageSetVersionString, RecordDescriptors, and FirmwareDevicePackageData fields.
uint8	**DescriptorCount** The number of descriptors included within the RecordDescriptors field for this record.

Individual Firmware Device ID Record (this section is repeated for each Firmware Device ID)	
Type	**Definition**
bitfield32	**DeviceUpdateOptionFlags** 32 bit field, each bit represents an update option. [31:1] – Reserved [0] – Continue component updates after failure If set, the UA shall attempt to update any remaining components after an individual component update fails as the FD will remain in the Update mode. This includes continuing after a non-zero ComponentResponseCode is received from the FD in the PassComponentTable command response.
enum8	**ComponentImageSetVersionStringType** The type of string used in the ComponentImageSetVersionString field. Refer to Table 20 for values.
uint8	**ComponentImageSetVersionStringLength** The length, in bytes, of the ComponentImageSetVersionString.
uint16	**FirmwareDevicePackageDataLength** The length in bytes of the FirmwareDevicePackageData field. If no data is provided in the firmware update package for the Firmware Device described by this portion of the header, then this length field should be set to 0x0000.
Variable Bitfield	**ApplicableComponents** The size of this bitfield is based on the value contained in the ComponentBitmapBitLengthfield. Bitmap of which firmware components are applicable to FDs that match this Device Identifier record. A set bit N indicates the Nth (0-based) component in the payload (which is described by the Nth entry in the component information area of the package header) is applicable to this device. Since the Component Bitmap Bit Length field (a multiple of 8) may contain bit positions not associated with any component (if the number of components is not a multiple of 8), those bit positions will contain 0 and are located in the high order bit positions within the bitfield.
Variable	**ComponentImageSetVersionString** Component Image Set version information, up to 255 bytes. Contains a variable type string describing the version of the set of component images that are applicable to the firmware device indicated in this device ID record.
Variable	**RecordDescriptors** Refer to Table 6 for details of these fields and the values that can be selected.
Variable	**FirmwareDevicePackageData** An optional data field that can be provided within the firmware update package that the UA shall transfer to the FD during the firmware update process. The UA has no knowledge of what data is contained within this field, and will simply pass the contents of this field when the FD requests it via the GetPackageData command response. If the FirmwareDevicePackageDataLength field is set to 0x0000 then this field contains no data and is zero bytes in length.

A firmware device record shall have at least one descriptor, but typically will have additional descriptors that the UA will use to match against a FD. Each descriptor is comprised of three fields: (1) Type (2) Length (3) Value. The initial descriptor is restricted to one of three types, while additional descriptors can choose from a larger range of type values including a vendor defined type. Refer to Table 6 for more details.

The contents of the ComponentImageInformation field is described in Table 5.

707 **Table 5 – Component image information**

Individual Component Image Information (repeated for each component image)	
Type	**Definition**
uint16	**ComponentClassification** FD vendor selected value to indicate specific FD component. Values for this field are aligned with the Value Map from CIM_SoftwareIdentify.Classifications. Refer to Table 19 for values.
uint16	**ComponentIdentifier** FD vendor selected unique value to distinguish between component images.
uint32	**ComponentComparisonStamp** When ComponentOptions bit 1 is set, this field shall contain a FD vendor selected value to use as a comparison value in determining if a firmware component is down-level or up-level. For the same component identifier, the greater of two component comparison stamps is considered up-level compared to the other when performing an unsigned integer comparison. FD vendors should choose the value for the comparison stamp in a manner that permits interim component versions such as patch releases. For example, a value for this field may follow the format of MajorMinorRevisionPatch where each subfield has a range of 0x00 to 0xFF. When ComponentOptions bit 1 is not set, this field should use the value of 0xFFFFFFFF.
bitfield16	**ComponentOptions** [15:2] – reserved [1] – Use Component Comparison Stamp When set, this bit indicates to the UA that the ComponentComparisonStamp field should be used for comparing this component against the component currently installed within the FD. If this bit is not set, the UA can only use the ComponentVersionString information that may not provide a direct comparison method to determine whether the component is higher or lower than one that is currently installed within the FD. [0] - Force Update When set, this bit indicates to the UA that it should request a comparison override (update the firmware component even if the update would take the component to a lower or equal component comparison stamp, or version string, than is currently active) in the UpdateComponent command for this component.
bitfield16	**RequestedComponentActivationMethod** Provides the ability for the firmware update package to request an activation method that the UA should use for the component images being updated. The UA would use the information from this field, along with the activation methods supported by the firmware device directly to determine the appropriate method for activation of the new code. Set each requested activation method to 1b (multiple choices are possible). [15:6] – Reserved [5] - AC power cycle [4] - DC power cycle [3] - System reboot [2] - Medium-specific reset [1] - Self-Contained (can be performed upon transmission of ActivateFirmware command) [0] - Automatic (becomes active as the Apply completes, or as download completes if the FD performs an auto-apply)
uint32	**ComponentLocationOffset** Offset in Bytes from byte 0 of the package header to where the component image begins.

Individual Component Image Information (repeated for each component image)	
Type	**Definition**
uint32	**ComponentSize** Size in Bytes of the Component image.
enum8	**ComponentVersionStringType** The type of string used in the ComponentVersionStringField. Refer to Table 20 for values.
uint8	**ComponentVersionStringLength** The length, in bytes, of the ComponentVersionString.
Variable	**ComponentVersionString** Component version information up to 255 bytes. Contains a variable type string describing the component version.

The content of the RecordDescriptors field is described in Table 6.

Table 6 – Descriptor definition

Initial Descriptor (This first initial descriptor (Type, Length, and Value) is mandatory)	
Type	**Definition**
uint16	**InitialDescriptorType** Indicates the type of the Initial descriptor. Refer to Table 7 for possible values. The initial descriptor for a device shall be defined by one of the following (PCI Vendor ID, IANA Enterprise ID, UUID, PnP Vendor ID, or ACPI Vendor ID). If the FD uses Vendor Defined values as part of its implementation of this specification (for example to provide a vendor defined error code or component classification), then the initial descriptor shall be set to either PCI Vendor ID or IANA Enterprise ID.
uint16	**InitialDescriptorLength** Indicates the length, in bytes, of the InitialDescriptorData field. Refer to Table 7 for possible values.
Variable	**InitialDescriptorData** Payload containing the identifier value for the initial descriptor. Refer to Table 7 for details.
Optional Additional Descriptors (repeated for each additional descriptor) **For each additional descriptor three fields are provided (Type, Length, Value)**	
Type	**Definition**
uint16	**AdditionalDescriptorType** Indicates the type of the additional descriptor. Refer to Table 7 for possible values.
uint16	**AdditionalDescriptorLength** Indicates the length, in bytes, of the AdditionalDescriptorIdentifierData field. Refer to Table 7 for possible values.
Variable	**AdditionalDescriptorIdentifierData** Payload containing the identifier value for the additional descriptors. Refer to Table 7 for details.

Table 7 provides a list of available descriptor types that can be used by the firmware package header and FD devices. When the FD is a PCI device, there are up to four descriptors that are mandatory to be implemented.

713 Table 7 – Descriptor identifier table

Any one of the highlighted rows can be used for the Initial Device Descriptor		
Type	**Length**	**Value**
0x0000 – PCI Vendor ID	2 bytes	PCI Vendor ID assigned to the FD vendor. If the FD is a PCI device, this descriptor shall be the initial descriptor.
0x0001 – IANA Enterprise ID	4 bytes	IANA Enterprise ID assigned to the FD vendor.
0x0002 – UUID	16 bytes	UUID assigned to the FD. Refer to PLDM Base Specification for UUID format. Version 1 format is recommended.
0x0003 – PnP Vendor ID	3 bytes	PnP Vendor ID, in ASCII characters, assigned to the FD vendor. Refer to the PnP & ACPI Registry for more details. http://www.uefi.org/PNP_ACPI_Registry
0x0004 – ACPI Vendor ID	4 bytes	ACPI Vendor ID, in ASCII characters, assigned to the FD vendor. Refer to the PnP & ACPI Registry for more details. http://www.uefi.org/PNP_ACPI_Registry
0x0100 – PCI Device ID	2 bytes	PCI Device ID assigned by the FD vendor. If the FD is a PCI device, this descriptor shall be provided in the QueryDeviceIdentifiers command response and shall also be used in the Descriptor Definition of the PLDM Firmware Packet Header.
0x0101 – PCI Subsystem Vendor ID	2 bytes	PCI Subsystem Vendor ID assigned to the FD vendor. If the FD is a PCI device, this descriptor shall be provided in the QueryDeviceIdentifiers command response. This descriptor can optionally be used in the Descriptor Definition of the PLDM Firmware Packet Header.
0x0102 – PCI Subsystem ID	2 bytes	PCI Subsystem Device ID assigned by the FD vendor. If the FD is a PCI device, this descriptor shall be provided in the QueryDeviceIdentifiers command response. This descriptor can optionally be used in the Descriptor Definition of the PLDM Firmware Packet Header.
0x0103 – PCI Revision ID	1 byte	PCI Revision ID assigned by the FD vendor.
0x0104 – PnP Product Identifier	4 bytes	PnP Product Identifier, in ASCII characters, assigned to the FD vendor. Refer to the PnP & ACPI Registry for more details. http://www.uefi.org/PNP_ACPI_Registry
0x0105 – ACPI Product Identifier	4 bytes	ACPI Product Identifier, in ASCII characters, assigned by the FD vendor. Refer to the PnP & ACPI Registry for more details. http://www.uefi.org/PNP_ACPI_Registry
0xFFFF – Vendor Defined	Variable	See Table 8 If the FD or package header uses a Vendor Defined value then the initial descriptor shall be set to either PCI Vendor ID or IANA Enterprise ID.

714 Table 8 provides details for the value field of a vendor-defined descriptor.

Table 8 – Vendor-defined descriptor value definition

Type	Definition
enum8	**VendorDefinedDescriptorTitleStringType** The type of string used in the VendorDefinedDescriptorTitleString field. Refer to Table 20 for values
uint8	**VendorDefinedDescriptorTitleStringLength** The length, in bytes, of the VendorDefinedDescriptorTitleString.
Variable	**VendorDefinedDescriptorTitleString** Vendor Defined Descriptor information up to 255 bytes. Contains a variable type string describing the Vendor's descriptor for the FD device.
Variable	**VendorDefinedDescriptorData** Vendor-specific descriptor value. Value will be treated as binary data by the UA.

7.1 Package to firmware device association

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

```
FOR each FD that supports PLDM for Firmware Update
    Retrieve FD identifier records via the QueryDeviceIdentifiers command
    MATCH = FALSE; Start at First Device Identifier Record in the package header
    WHILE ((MATCH==FALSE) AND (Device Identifier Record(s) remain in package))
        Read Device Identifier Record from Package Header
        IF all Package Device Identifier Record descriptors match FD descriptors
            MATCH = TRUE; Selected Record = Current Record; Break;
        Move to next Device Identifier Record in package header
```

Note that all descriptors in a package Device Identifier Record shall match those returned by the FD but not vice-versa (the FD may return more descriptors than are indicated in the firmware package header Device Identifier record).

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

8 Operational behaviors

This clause describes the operating states of the FD.

8.1 State definitions

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

- IDLE

  IDLE is the default state in which the firmware device shall always start after an initialization. In this state the FD is not performing any firmware update actions as it has not received a RequestUpdate command from the UA.

- **LEARN COMPONENTS**

  After receiving the RequestUpdate command, the FD moves to this state while waiting to receive the PassComponentTable command from the UA. The FD will then learn the size, identifier, component comparison stamp, classification and version of the component images the UA intends to send.

- **READY XFER**

  After learning the component image information, the FD moves to this state to wait for the command initiating a component image transfer. This state is re-entered after each component image is transferred, verified and applied. The FD remains in this state after all firmware components have been applied as it waits for an activation command.

- **DOWNLOAD**

  After receiving the command to update a firmware component, the FD moves to this state to begin requesting the transfer of portions of the component image from the UA. When an entire component image has been transferred, the UA is informed and the FD moves to the VERIFY state.

- **VERIFY**

  In this state the FD performs a validation check of the firmware component; it is up to the FD to determine the method used for verification of the code image. Upon successful verification, the FD informs the UA and moves to the APPLY state.

- **APPLY**

  In this state the FD writes the verified code image to the non-volatile storage area that will contain the code image within the device. When completed, the FD moves to the READY XFER state

- **ACTIVATE**

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

8.2 State machine

Table 9 describes the operating states, responses, and transitions between states that the FD shall implement. The transition to the next state occurs after the FD performs the response action. In cases where the FD is sending a command to the UA, the transition does not occur until the UA successfully acknowledges the command (i.e., with a corresponding response and CompletionCode value of 0). Two commands, GetFirmwareParameters and QueryDeviceIdentifiers, are considered ‘inventory’ type commands and can be sent by the UA to the FD in any state. In addition, the GetStatus command may also be sent from the UA to the FD in any state.

777

778 **Table 9 – Firmware device state machine**

Current State	Trigger	Response	Next State
IDLE	RequestUpdate	Success	LEARN COMPONENTS
	RequestUpdate	Unable to Initiate Update or Retry Request Update	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	IDLE
	GetFirmwareParameters	Success with firmware info	IDLE
	GetStatus	Success with info	IDLE
	Any other command	Not in Update Mode	IDLE
LEARN COMPONENTS	FD_T1 timeout waiting for next command or response to GetPackageData	None	IDLE
	GetPackageData	Success	LEARN COMPONENTS
	GetDeviceMetaData	Success	LEARN COMPONENTS
	PassComponentTable with valid TransferFlag set to Start or Middle	Success	LEARN COMPONENTS
	PassComponentTable with valid TransferFlag set to End or StartAndEnd	Success	READY XFER
	PassComponentTable with invalid TransferFlag	Error CompletionCode	LEARN COMPONENTS
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	LEARN COMPONENTS
	GetFirmwareParameters	Success with firmware info	LEARN COMPONENTS
	GetStatus	Success with info	LEARN COMPONENTS
	Any other Update command	Invalid State Machine	LEARN COMPONENTS

Current State	Trigger	Response	Next State
READY XFER	FD_T1 timeout waiting for next command	None	IDLE
	RequestUpdate	Already In Update Mode	READY XFER
	GetFirmwareParameters	Success with firmware info	READY XFER
	UpdateComponent with invalid or unsupported parameters	Non-zero ComponentCompatibilityResponseCode response	READY XFER
	UpdateComponent with supported and acceptable parameters	Success	DOWNLOAD
	GetMetaData	Success	READY XFER
	ActivateFirmware after all expected components have completed transfer, verify and apply	Success with Activation Delay Time	ACTIVATE
	ActivateFirmware prior to all expected components completed	Incomplete Update response	READY XFER
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	READY XFER
	GetFirmwareParameters	Success with firmware info	READY XFER
	GetStatus	Success indicating READY XFER state	READY XFER
	Any other Update command	Invalid State Machine	READY XFER

Current State	Trigger	Response	Next State
DOWNLOAD	FD_T1 timeout waiting for response to RequestFirmwareData	None	IDLE
	Ready to request next component image portion	Send RequestFirmwareData command	DOWNLOAD
	Receive RequestFirmwareData response with image portion	Process data	DOWNLOAD
	All necessary data received and processed for this component	Send TransferComplete command with succesful TransferResult	VERIFY
	Corrupt data received	Send TransferComplete command with failure TransferResult	DOWNLOAD
	Error response to RequestFirmwareData	Send TransferComplete command with failure TransferResult	DOWNLOAD
	Retry response to RequestFirmwareData	Delay, then send RequestFirmwareData command for same component image portion as prior request)	DOWNLOAD
	CancelUpdateComponent	Success	READY XFER
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	DOWNLOAD
	GetFirmwareParameters	Success with firmware info	DOWNLOAD
	GetMetaData	Success	DOWNLOAD
	GetStatus while downloading	Download in progress	DOWNLOAD
	GetStatus after successful download	Download successful	DOWNLOAD
	Any other command	Invalid State Machine	DOWNLOAD
VERIFY	GetStatus while verifying	Verification in progress	VERIFY
	GetStatus after successful verify	Verification successful	VERIFY
	GetStatus after failure to verify	Verification failed	VERIFY
	Verify completes successfully	Send VerifyComplete command with successful VerifyResult	APPLY
	Verify ended with failure	Send VerifyComplete command with failure VerifyResult	VERIFY
	CancelUpdateComponent	Success	READY XFER
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	VERIFY
	GetFirmwareParameters	Success with firmware info	VERIFY
	GetMetaData	Success	VERIFY
	Any other command	Invalid State Machine	VERIFY

Current State	Trigger	Response	Next State
APPLY	GetStatus while applying	Apply in progress	APPLY
	GetStatus after successful apply	Apply successful	APPLY
	GetStatus after apply failure	Apply failed	APPLY
	Apply completes successfully	Send ApplyComplete command with successful ApplyResult	READY XFER
	Apply ended with failure	Send ApplyComplete command with failure ApplyResult	APPLY
	CancelUpdateComponent	Success	READY XFER
	CancelUpdate	Success	IDLE
	QueryDeviceIdentifiers	Success with Identifiers	APPLY
	GetFirmwareParameters	Success with firmware info	APPLY
	GetMetaData	Success	APPLY
	Any other command	Invalid State Machine	APPLY
ACTIVATE	Sets transferred component image to become active firmware component on next activation	Success	IDLE
	Self-contained activation option is requested for applicable components	Success with maximum activation time in response	ACTIVATE
	Self-contained activation completes	Idle state	IDLE
	GetStatus	Activate state	ACTIVATE
	QueryDeviceIdentifiers	Success with Identifiers	ACTIVATE
	GetFirmwareParameters	Success with firmware info	ACTIVATE
	GetMetaData	Success	ACTIVATE
	Any other command	Invalid State Machine	ACTIVATE

8.3 State transition diagram

The diagram in Figure 6 illustrates the state transitions the FD shall implement. Each bubble represents a particular state as defined in Table 9. Upon initialization, system reboot, or a device reset the FD shall enter the IDLE state. The dashed lines represent state change transitions, not due to timeouts, which are initiated by the FD while the solid lines indicate transitions that are initiated by the UA.

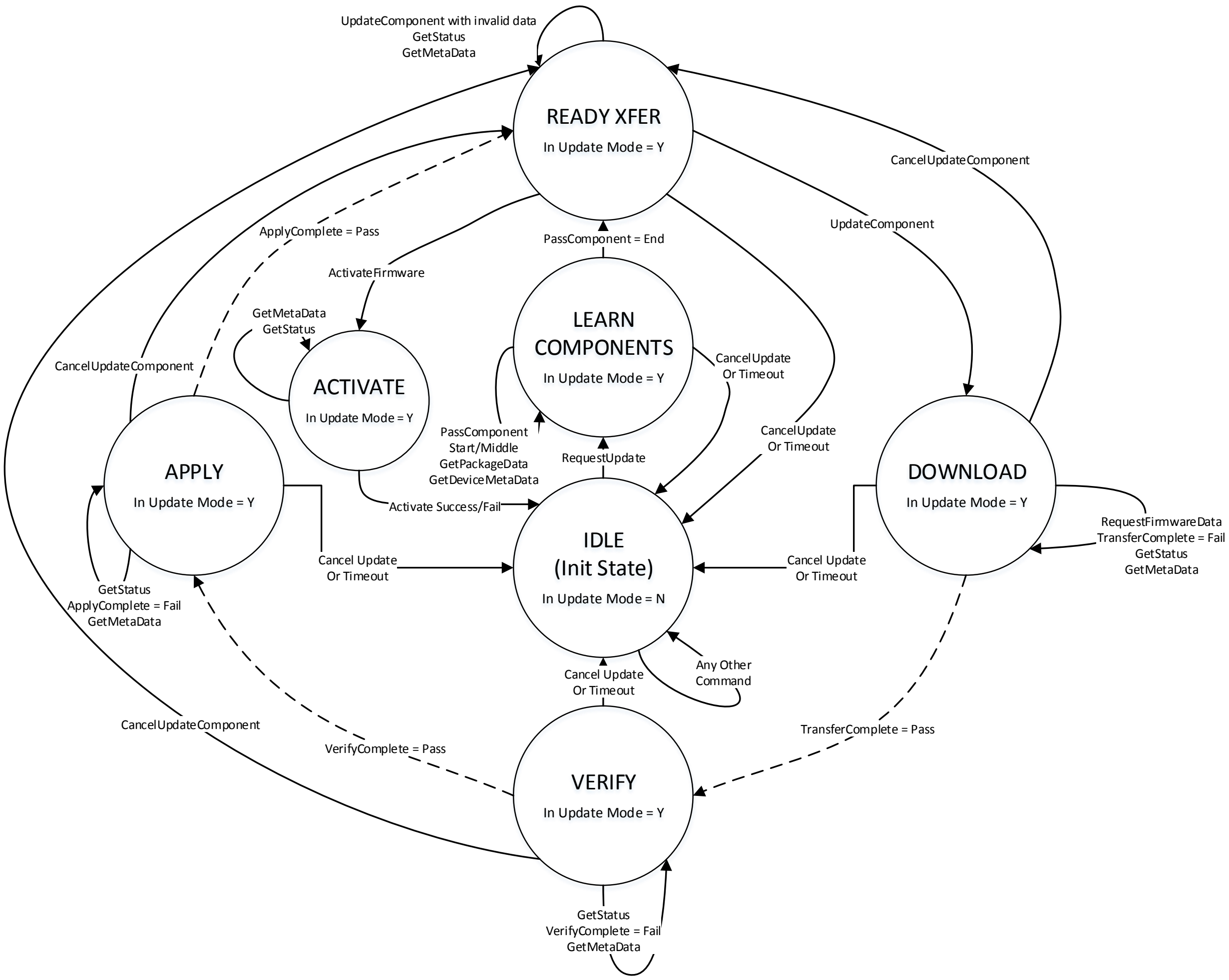

Figure 6 – Firmware device state transition diagram

9 PLDM commands for firmware update

This clause provides the list of command codes that are used by Update Agents and Firmware Devices that implement PLDM Firmware Updates as defined in this specification. The command codes for the PLDM messages are given in Table 10.

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

Table 10 – PLDM for firmware update command codes

Command	Command Code	Command Requirement for UA	Command Requirement for FD		Command Requestor (Initiator)	Reference
			FD implementing full update capability	FD implementing inventory only support		
INVENTORY COMMANDS						
QueryDeviceIdentifiers	0x01	Mandatory	Mandatory	Mandatory	UA	See 10.1
GetFirmwareParameters	0x02	Mandatory	Mandatory	Mandatory	UA	See 10.2
Reserved	0x03-0x0F					
UPDATE COMMANDS						
RequestUpdate	0x10	Mandatory	Mandatory	Optional	UA	See 11.1
GetPackageData	0x11	Mandatory	Optional	Optional	FD	See 11.2
GetDeviceMetaData	0x12	Mandatory	Optional	Optional	UA	See 11.3
PassComponentTable	0x13	Mandatory	Mandatory	Optional	UA	See 11.4
UpdateComponent	0x14	Mandatory	Mandatory	Optional	UA	See 11.5
RequestFirmwareData	0x15	Mandatory	Mandatory	Optional	FD	See 11.6
TransferComplete	0x16	Mandatory	Mandatory	Optional	FD	See 11.7
VerifyComplete	0x17	Mandatory	Mandatory	Optional	FD	See 11.8
ApplyComplete	0x18	Mandatory	Mandatory	Optional	FD	See 11.9
GetMetaData	0x19	Mandatory	Optional	Optional	FD	See 11.10
ActivateFirmware	0x1A	Mandatory	Mandatory	Optional	UA	See 11.11
GetStatus	0x1B	Mandatory	Mandatory	Optional	UA	See 11.12
CancelUpdateComponent	0x1C	Mandatory	Mandatory	Optional	UA	See 11.13
CancelUpdate	0x1D	Mandatory	Mandatory	Optional	UA	See 11.14

10 PLDM for firmware update – inventory commands

This clause describes the commands that are used by Update Agents and Firmware Devices that implement the inventory commands that are defined in this specification. The command codes for the PLDM messages are given in Table 10.

10.1 QueryDeviceIdentifiers command format

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

Table 11 – QueryDeviceIdentifiers command format

Type	**Request data**
--	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES }
uint32	**DeviceIdentifiersLength** Contains the length, in bytes, of the Descriptors field.
uint8	**DescriptorCount** The total number of descriptors for the FD device.
Variable	**Descriptors** Refer to Table 6 for details on the format and values for these fields.

10.2 GetFirmwareParameters command format

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

Table 12 – GetFirmwareParameters command format

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

Type	Response data
bitfield32	**CapabilitiesDuringUpdate** 32 bit field, specifying the capability of the firmware device. Bit [31:8] – Reserved Bit [7:4] – Firmware Device Update Mode Restrictions Bit 4: 0 – No host OS environment restriction for update mode 1 – Firmware device unable to enter update mode if host OS environment is active. Bit 7:5 -- Reserved Bit [3] – Firmware Device Partial Updates 0: Firmware Device cannot accept a partial update and all components present on the FD shall be updated. 1: Firmware Device can support a partial update, whereby a package that contains a component image set that is a subset of all components currently residing on the FD, can be transferred. Note: The UA shall always transfer the entire component image set provided by the firmware update package. No provision is defined within this specification that would allow a UA to only transfer a portion of the component image set. Bit [2] – Firmware Device Host Functionality during Firmware Update 0: Device host functionality is not reduced during Firmware Update. 1: Device host functionality will be reduced, perhaps becoming inaccessible, during Firmware Update. Bit [1] – Component Update Failure Retry Capability 0: Device can have component updated again without exiting update mode and restarting transfer via RequestUpdate command. 1: Device will not be able to update component again unless it exits update mode and the UA sends a new Request Update command. Bit [0] – Component Update Failure Recovery Capability 0: Device will revert to previous component image upon a failure, timeout, or cancelation of the transfer. 1: Device will not revert to previous component image upon a failure, timeout, or cancelation of the transfer. Therefore the current pending component version may be corrupt if the transfer does not complete.
uint16	**ComponentCount** Number of firmware components that reside within the FD. Each one will have an entry in the following ComponentParameterTable.
enum8	**ActiveComponentImageSetVersionStringType** The type of string used in the ActiveComponentImageSetVersionString field. Refer to Table 20 for values.
uint8	**ActiveComponentImageSetVersionStringLength** The length, in bytes, of the ActiveComponentImageSetVersionString.

Type	Response data
enum8	**PendingComponentImageSetVersionStringType** The type of string used in the PendingComponentImageSetVersionString field. This field, and all other pending component image set fields, are valid once the firmware device has received the ActivateFirmware command to prepare the firmware components for activation, but the activation method requires further action to enable the pending images to become the actively running code images. Refer to Table 20 for values. If no pending component image set exists, this value shall be set to '0 – Unknown'.
uint8	**PendingComponentImageSetVersionStringLength** The length, in bytes, of the PendingComponentImageSetVersionString. Refer to PendingComponentImageSetVersionStringType field for additional details. If no pending component image set exists, this value shall be set to 0x0.
Variable	**ActiveComponentImageSetVersionString** Component Image Set version information, up to 255 bytes. Contains a variable type string describing the version of the set of component images that are currently active.
Variable	**PendingComponentImageSetVersionString** Component image set version, which is pending activation, up to 255 bytes. The version reported here should be the one that will become active on the next initialization or activation of the components. The pending component image set version value may be same as the active component image set version. Contains a variable type string describing the pending component image set version. Refer to PendingComponentImageSetVersionStringType field for additional details. If no pending component image set exists, this field is zero bytes in length.
Variable	**ComponentParameterTable** Table of component entries for all of the updateable components that reside on the FD. Refer to Table 13 for details.

811 **Table 13 – ComponentParameterTable -- entry format**

Type	Data
uint16	**ComponentClassification** Vendor specific component classification information. Refer to Table 19 for specific values. Special values: 0x0000, 0xFFFF = reserved.
uint16	**ComponentIdentifier** FD vendor selected unique value to distinguish between component images.
uint8	**ComponentClassificationIndex** Used to distinguish identical components that have the same classification and identifier that can use the same component image but the images are stored in different locations in the FD.
uint32	**ActiveComponentComparisonStamp** Optional Firmware component comparison stamp that is currently active. If the firmware component does not provide a component comparison stamp, this value should be set to 0x00000000.
enum8	**ActiveComponentVersionStringType** The type of strings used in the ActiveComponentVersionString field. Refer to Table 20 for values.

Type	Data
uint8	**ActiveComponentVersionStringLength** The length, in bytes, of the ActiveComponentVersionString.
ASCII[8]	**ActiveComponentReleaseDate** Eight byte field containing the date corresponding to the component version level being reported – Format YYYYMMDD. If the firmware component does not provide a date, this value shall be set to ASCII null characters represented by eight 0x00 bytes.
uint32	**PendingComponentComparisonStamp** Optional firmware component comparison stamp that is pending activation. This field, and all other pending component fields, are valid once the firmware device has received the ActivateFirmware command to prepare the firmware component for activation, but the activation method requires further action to enable the pending image to become the actively running code image. If no pending firmware component exists, this value shall be set to 0x00000000.
enum8	**PendingComponentVersionStringType** The type of strings used in the PendingComponentVersionString field. Refer to PendingComponentComparisonStamp field for additional details. Refer to Table 20 for values. If no pending Firmware Component exists, this value shall be set to ‘0 – Unknown’.
uint8	**PendingComponentVersionStringLength** The length, in bytes, of the PendingComponentVersionString. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this value shall be set to 0x0.
ASCII[8]	**PendingComponentReleaseDate** Eight byte field containing the date corresponding to the component version level being reported – Format YYYYMMDD. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this value shall be set to ASCII null characters represented by eight 0x00 bytes
bitfield16	**ComponentActivationMethods** Provides the capability of the FD for firmware activation. [15:6] – reserved [5] - AC power cycle [4] - DC power cycle [3] - System reboot [2] - Medium-specific reset [1] - Self-Contained (can be performed upon transmission of ActivateFirmware command) [0] - Automatic (becomes active as the Apply completes, or as download completes if the FD performs an auto-apply)

Type	Data
Bitfield 32	**CapabilitiesDuringUpdate** 32 bit field, containing capability of the firmware component. Bit [31:1] – Reserved Bit [0] – Firmware Device apply state functionality. 0: Firmware Device will execute an operation during the APPLY state that will include migrating the new component image to its final non-volatile storage destination. 1: Firmware Device performs an 'auto-apply' during transfer phase and apply step will be completed immediately.
Variable	**ActiveComponentVersionString** Firmware component version, which is currently active, up to 255 bytes. Contains a variable type string describing the active component version.
Variable	**PendingComponentVersionString** Firmware component version, which is pending activation, up to 255 bytes. The version reported here should be the one that will become active on the next initialization or activation of the component. The pending component version value may be same as the active component version. Contains a variable type string describing the pending component version. Refer to PendingComponentComparisonStamp field for additional details. If no pending firmware component exists, this field is zero bytes in length.

11 PLDM for firmware update – update commands

This clause describes the commands that are used by Update Agents and Firmware Devices that implement the firmware update capability as defined in this specification. The command numbers for the PLDM messages are given in Table 10.

11.1 RequestUpdate command format

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

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

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

Table 14 -- RequestUpdate command format

Type	Request data
uint32	**MaximumTransferSize** Specifies the maximum size, in bytes, of the variable payload allowed to be requested by the FD via the RequestFirmwareData command that is contained within a PLDM message. This value shall be equal to or greater than firmware update baseline transfer size. Refer to clause 6.6 for details on the firmware update baseline transfer size.
uint16	**NumberOfComponents** Specifies the number of components that will be passed to the FD during the update. The FD can use this value to compare against the number of PassComponentTable commands received.

Type	Request data
uint8	**MaximumOutstandingTransferRequests** Specifies the number of outstanding RequestFirmwareData commands that can be sent by the FD. The minimum required value is ‘1’, which the UA shall support. It is optional for the UA to support a value higher than ‘1’ for this field.
uint16	**PackageDataLength** This field shall be set to the value contained within the FirmwareDevicePackageDataLength field that was provided in the firmware package header. If no firmware package data was provided in the firmware update package then this length field shall be set to 0x0000.
enum8	**ComponentImageSetVersionStringType** The type of string used in the ComponentImageSetVersionString field. Refer to Table 20 for values.
uint8	**ComponentImageSetVersionStringLength** The length, in bytes, of the ComponentImageSetVersionString.
Variable	**ComponentImageSetVersionString** Component Image Set version information, up to 255 bytes. Contains a variable type string describing the version of the set of component images that will be transferred to the FD.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, ALREADY_IN_UPDATE_MODE, UNABLE_TO_INITIATE_UPDATE, RETRY_REQUEST_UPDATE }
uint16	**FirmwareDeviceMetaDataLength** This field shall be set to the length of the metadata that the FD needs the UA to retain during the firmware update process. If the firmware device has no metadata to be retained during the firmware update process then this length field shall be set to 0x0000.
uint8	**FDWillSendGetPackageDataCommand** Set to 0x01 if the PackageDataLength field indicated that there was package data that the FD should obtain, and the FD will request this data at the beginning of the learn components state. Set to 0x00 if the PackageDataLength field was 0x0000, or if there was package data but the FD does not support the optional GetPackageData command. All other values reserved

Error completion codes handling:

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

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

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

11.2 GetPackageData command format

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

Table 15 – GetPackageData command format

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

Error completion codes handling:

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

- NO_PACKAGE_DATA: Returned from the UA if there is no firmware package data that needs to be sent to the FD.

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

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

11.3 GetDeviceMetaData command format

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

Table 16 – GetDeviceMetaData command format

Type	Request data
uint32	**DataTransferHandle** A handle that is used to identify a package data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart.
enum8	**TransferOperationFlag** The operation flag that indiates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01}
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_STATE_FOR_COMMAND, NO_DEVICE_METADATA, INVALID_TRANSFER_HANDLE, INVALID_TRANSFER_OPERATION_FLAG }
uint32	**NextDataTransferHandle** A handle that is used to identify the next portion of the transfer.
enum8	**TransferFlag** The transfer flag that indiates what part of the transfer this response represents. Possible values: {Start=0x01, Middle=0x02, End=0x04, StartAndEnd=0x05}
Variable	**PortionOfMetaData** A portion of the firmware device metadata that the UA shall obtain and retain during the firmware update process.

Error completion codes handling:

- INVALID_STATE_FOR_COMMAND: The FD only expects this command in LEARN COMPONENTS state.
- NO_DEVICE_METADATA: Returned from the FD if there is no metadata that needs to be transferred to the UA.
- INVALID_TRANSFER_HANDLE: Returned from the FD if the transfer handle used in the request is invalid.
- INVALID_TRANSFER_OPERATION_FLAG: Returned from the FD if the transfer operation flag is invalid

11.4 PassComponentTable command format

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

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

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

Table 17 – PassComponentTable command format

Type	Request data
enum8	**TransferFlag** The transfer flag that indicates what part of the Component Table this request represents. Possible values: {Start = 0x1, Middle = 0x2, End = 0x4, StartAndEnd = 0x5}
uint16	**ComponentClassification** Vendor specific component classification information. Refer to Table 19 for specific values. Special values: 0x0000, 0xFFFF = reserved.
uint16	**ComponentIdentifier** FD vendor selected unique value to distinguish between component images.
uint8	**ComponentClassificationIndex** The component classification index that was obtained from the GetFirmwareParameters command to indicate which firmware component the information contained within this command is applicable for.
uint32	**ComponentComparisonStamp** FD vendor selected value to use as a comparison value in determining if a firmware component is down-level or up-level. For the same component identifier, the greater of two component comparison stamps is considered up-level compared to the other when performing an unsigned integer comparison.
enum8	**ComponentVersionStringType** The type of strings used in the ComponentVersionString field. Refer to Table 20 for values.
uint8	**ComponentVersionStringLength** The length, in bytes, of the ComponentVersionString.
Variable	**ComponentVersionString** Firmware component version information up to 255 bytes. Contains a variable type string describing the component version.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE, INVALID_STATE_FOR_COMMAND }
enum8	**ComponentResponse** The FD should reply back with initial compatibility with component provided by UA. 0 – Component can be updated – ComponentResponseCode shall be set to 0x00. 1 – Component may be updateable – A ComponentResponseCode greater than zero shall be provided to explain the reason why the component cannot be updated, or if a flag is required to be set in UpdateOptionFlags field within the UpdateComponent request. All other values reserved.

Type	Response data
uint8	**ComponentResponseCode** 0x00: Component can be updated. 0x01: Component comparison stamp is identical to the firmware component comparison stamp in the FD. Force update option flag (if supported by FD) will need to be set in the UpdateComponent request. 0x02: Component comparison stamp is lower than the firmware component comparison stamp in the FD. Force update option flag (if supported by FD) will need to be set to in the UpdateComponent request. 0x03: Invalid component comparison stamp. 0x04: Component has conflict with another component provided in a separate PassComponentTable command. 0x05: Pre-requisites for this component have not been met. 0x06: Component is not supported on FD. 0x07: Security restrictions prevent component from being downgraded. 0x08: Incomplete component image set was received. The FD will reject each UpdateComponent command with response code of 0x08. 0x09: Reserved 0x0A: Component version string is identical to the firmware component version string in the FD. Force update option flag (if supported by FD) will need to be set in the UpdateComponent request. This response code can be used only when component comparison stamp is not supported by the FD. 0x0B: Component version string is lower to the firmware component version string in the FD. Force update option flag (if supported by FD) will need to be set in the UpdateComponent request. This response code can be used only when component comparison stamp is not supported by the FD. 0x0C – 0xCF - Reserved 0xD0-0xEF: Firmware Device Vendor defined component response code. When an FD device uses a vendor defined status code, it shall also provide its Vendor ID information by using either the PCIe or IANA Vendor descriptor type. For details refer to Table 7. 0xF0 – 0xFF - Reserved

Error completion code handling:

- NOT_IN_UPDATE_MODE: Returned by the FD if it’s not currently in update mode.
- INVALID_STATE_FOR_COMMAND: The FD only expects this command in LEARN COMPONENTS state.

11.5 UpdateComponent command format

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

894 **Table 18 – UpdateComponent command format**

Type	Request data
uint16	**ComponentClassification** Classification value provided by the firmware package header information for the component to be transferred. Values for this field are aligned with the Value Map from CIM_SoftwareIdentity.Classifications. Refer to Table 19 for values.
uint16	**ComponentIdentifier** FD Vendor selected unique value to distinguish between component images.
uint8	**ComponentClassificationIndex** The component classification index that was obtained from the GetFirmwareParameters command to indicate which firmware component should be updated.
uint32	**ComponentComparisonStamp** FD vendor selected value to use as a comparison value in determining if a firmware component is down-level or up-level. For the same component identifier, the greater of two component comparison stamps is considered up-level compared to the other when performing an unsigned integer comparison.
uint32	**ComponentImageSize** Size in bytes of the component image.
bitfield32	**UpdateOptionFlags** 32 bits field, where each non-reserved bit represents an update option that can be requested by the UA to be enabled for the transfer of this component image. [31:1] – reserved [0] – Request Force Update of component – Can be used to inform the FD device to perform a transfer even if the component has a lower or equal component comparison stamp, or version string, than what is currently installed. The UA will set this bit for any component that has the force update bit set in the ComponentOptions field of the package header. Additionally, the UA could set the bit as instructed by commands used to provide the update package to the UA (these commands are out of scope for this spec).
enum8	**ComponentVersionStringType** The type of strings used in the ComponentVersionString field. Refer to Table 20 for values.
uint8	**ComponentVersionStringLength** The length, in bytes, of the ComponentVersionString.
Variable	**ComponentVersionString** Firmware component version information up to 255 bytes. Contains a variable type string describing the component version.

Type	Response data
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE}
enum8	**ComponentCompatibilityResponse** The FD should reply back with initial compatibility with component provided by UA. 0 – Component can be updated, and the FD will begin to request data via the RequestFirmwareData command. ComponentCompatibilityResponseCode shall be set to 0x00. 1 – Component will not be updated, and the FD will not begin to request component image data. A ComponentCompatibilityResponseCode greater than zero shall be provided to explain the reason for the FD rejection of the component. All other values reserved.
uint8	**ComponentCompatibilityResponse Code** 0x00: No response code – used when component can be updated. 0x01: Component comparison stamp is identical to the firmware component comparison stamp in the FD, but force update flag is not set. Force update option flag (if supported by FD) will need to be set to update component. Can also be used if FD does not support force flag. 0x02: Component comparison stamp is lower than the firmware component comparison stamp in the FD, but force update flag is not set. Force update option flag (if supported by FD) will need to be set to update component. Can also be used if FD does not support force flag. 0x03: Invalid component comparison stamp or version. 0x04: Component has conflict with another component provided in a separate PassComponentTable command. 0x05: Pre-requisites for this component have not been met. 0x06: Component is not supported on FD. 0x07: Security restrictions prevent component from being downgraded. Can be used when force update flag is set, but the firmware component cannot be downgraded. 0x08: Component cannot be updated as an Incomplete Component Image Set was received from the PassComponentTable commands. 0x09: Component information does not match details presented from PassComponentTable commands. 0x0A: Component version string is identical to the firmware component version string in the FD, but force update flag is not set. Force update option flag (if supported by FD) will need to be set to update component. Reason code can be used only when component comparison stamp is not supported by the FD. 0x0B: Component version string is lower to the firmware component version string in the FD, but force update flag is not set. Force update option flag (if supported by FD) will need to be set to update component. Reason code can be used only when component comparison stamp is not supported by the FD. 0x0C – 0xCF - Reserved 0xD0-0xEF: Firmware Device Vendor defined component response code. When an FD device uses a vendor defined status code, it shall also provide its Vendor ID information by using either the PCIe or IANA Vendor descriptor type. For details refer to Table 7**.** 0xF0 – 0xFF – Reserved

Type	Response data
bitfield32	**UpdateOptionFlagsEnabled** 32 bits field, where each non-reserved bit represents an update option that has been enabled by the FD for the transfer of this component image. This field provides the response from the FD to the request made by the UA in the UpdateOptionFlag field A '1' in the bit indicates the requested update option flag was accepted. [31:1] – Reserved [0] – Force Update of component; FD will perform a force update of the component.
uint16	**EstimatedTimeBeforeSendingRequestFirmwareData** Amount of time the FD requires to prepare before sending the first RequestFirmwareData command. Measured in seconds. If this field contains a non-zero value, the UA should not begin any of the timers listed in Table 2 until after the amount of time present in this field has elapsed. It is permissible for the FD to begin sending the RequestFirmwareData commands prior to when the timer would have elapsed.

895

896 **Table 19 – ComponentClassification values**

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

897

Table 20 – String type values

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

Error completion codes handling:

- NOT_IN_UPDATE_MODE: Returned by the FD if it’s not currently in update mode.

11.6 RequestFirmwareData command format

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

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

The FD can request the same data more than one time if it wants to perform an immediate verification of the data. The UA shall allow the FD to request data at any valid offset within the firmware data.

Table 21 – RequestFirmwareData command format

Type	Request data
uint32	**Offset** Offset of the component image segment within the current component being transferred.
uint32	**Length** Size of the component image segment requested by the FD. This value shall be set between the firmware update baseline transfer size, and the MaximumTransferSize value from the RequestUpdate command. Refer to clause 6.6 for details on the firmware update baseline transfer size.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, INVALID_TRANSFER_LENGTH, COMMAND_NOT_EXPECTED, DATA_OUT_OF_RANGE, RETRY_REQUEST_FW_DATA, CANCEL_PENDING }

Type	Response data
Variable	**ComponentImagePortion** The payload contains the portion corresponding to the component image from Offset to (Offset + Length – 1). The UA shall pad with 00s if the length requested extends past the end of the component image. The maximum amount of padding the UA shall support is equal to the firmware update baseline transfer size. Any request from the FD that would require a larger amount of pad bytes shall have its completion code set to DATA_OUT_OF_RANGE and no data is returned. Refer to clause 6.6 for details on the firmware update baseline transfer size. The permitted range of this ComponentImagePortion can be described by the following two equations: • Firmware Update Baseline Transfer Size <= Length <= MaximumTransferSize If this equation is not satisfied the UA shall return INVALID_TRANSFER_LENGTH • Offset + Length <= ComponentImageSize + Firmware Update Baseline Transfer Size If this equation is not satisfied the UA shall return DATA_OUT_OF_RANGE The maximum amount of pad bytes is equal to the firmware update baseline transfer size and can be described by the following equation: • Pad Bytes = Offset + Length – ComponentImageSize Below is an example of three request/responses each of that are within the permitted range for the ComponentImagePortion. ComponentImageSize = 160 bytes MaximumTransferSize = 512 bytes FD uses Length = 64 bytes Request #1 Offset = 0, Length = 64 Response #1 UA returns 64 bytes (Offset 0-63) from component image Request #2 Offset = 64, Length = 64 Response #2 UA returns 64 bytes (Offset 64-127) from component image Request #3 Offset = 128, Length = 64 Response #3 UA returns 32 bytes (Offset 128-159) from component image and 32 pad bytes of 0x00

Error completion codes handling:

- INVALID_TRANSFER_LENGTH: The length of the requested component image portion exceeds the MaxTransferSize in the RequestUpdate command, or is less than the firmware update baseline transfer size.
- COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not expected based on the sequence defined to update a firmware component.

- DATA_OUT_OF_RANGE: The requested component image portion offset exceeds the range of the component image, or would require the UA to pad the response with a number of bytes that is larger than the firmware update baseline transfer size. The FD can send another RequestFirmwareData command to attempt a retry with a different offset and length value.
- RETRY_REQUEST_FW_DATA: The requested component image portion is not currently available from the UA. The UA requests that the firmware device retry this command after FD_T2 as it may be retrieving the component image data from an external source.
- CANCEL_PENDING: The requested component image portion is not returned by the UA as it previously sent a CancelUpdate or CancelUpdateComponent command to the FD.

11.7 TransferComplete command format

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

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

Table 22 – TransferComplete command format

Type	Request data
uint8	**TransferResult** Use to indicate the result of the Download stage: 0x00: Transfer has completed without error. 0x01: Reserved 0x02: Transfer has completed with error as the version of the image received does not match the version expected from the UpdateComponent command. 0x03: Firmware Device has aborted the transfer. 0x04 - 0x08: Reserved 0x09: Timeout occurred while performing action. 0x0A: Generic Error has occurred. 0x0B – 0x6F: Reserved 0x70 – 0x8F: Firmware Device Vendor defined status code. When an FD device uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor descriptor type. For details refer to Table 4. 0x90 – 0xFF: Reserved When the FD has a result where multiple choices may be applicable, it should look to provide the most descriptive result code, which is applicable, in this field.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED}

Error completion codes handling:

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

11.8 VerifyComplete command format

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

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

After the FD finishes verifying the component successfully (including that the image data represents the expected version that was to be transferred), it issues the VerifyComplete command and transitions to the APPLY state. If the verification fails, the FD shall remain in the VERIFY state and issue VerifyComplete command indicating failed status of the verification. The UA shall send a CancelUpdateComponent command if a verification failure occurs

Table 23 – VerifyComplete command format

Type	Request data
uint8	**VerifyResult** Use to indicate the result of the Verify stage: 0x00: Verify has completed without error. 0x01: Verify has completed with a verification failure – FD will not transition to APPLY state to apply the component. 0x02: Verify has completed with error as the version of the image received does not match the version expected from the UpdateComponent command. – FD will not transition to APPLY state to apply the component. 0x03 - 0x08: Reserved 0x09: Timeout occurred while performing action – FD will not transition to APPLY state to apply the component. 0x0A: Generic Error has occurred – FD will not transition to APPLY state to apply the component. 0x0B – 0x8F: Reserved 0x90 - 0xAF: Firmware Device Vendor defined status code. When an FD device uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor define type. For details refer to Table 4. 0xB0 – 0xFF: Reserved When the FD has a result where multiple choices may be applicable, it should look to provide the most descriptive result code, which is applicable, in this field.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED }

Error completion codes handling:

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

11.9 ApplyComplete command format

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

Based on the newly applied component, if the FD determines that the activation method is different than what would be reported in the GetFirmwareParameters command prior to the component update, then the FD can set the appropriate bits in the ComponentActivationMethodsModification field.

Table 24 – ApplyComplete command format

Type	Request data
uint8	**ApplyResult** Used to indicate the result of the Apply stage: 0x00: Apply has completed without error. 0x01: Apply has completed with success and has modified its activation method. Values shall be provided in the ComponentActivationMethodsModifications field. 0x02: Apply has completed with a failure due to a memory write issue. 0x03 - 0x08: Reserved 0x09: Timeout occurred while performing action. 0x0A: Generic Error has occurred. 0x03 – 0xAF: Reserved 0xB0 – 0xCF: Firmware Device Vendor defined status code. When an FD device uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor define type. For details refer to Table 4. 0xD0 – 0xFF: Reserved When the FD has a result where multiple choices may be applicable, it should look to provide the most descriptive result code, which is applicable, in this field.
bitfield16	**ComponentActivationMethodsModification** Field contains a values when the ApplyResult is set to 0x01. Otherwise, each bit shall be set to ‘0’ Provides the capability of the FD for firmware activation. This supersedes the values provided by the FD via the GetFirmwareParameters command. [15:6] – Reserved [5] - AC power cycle [4] - DC power cycle [3] - System reboot [2] - Medium-specific reset [1] - Self-Contained (can be performed upon transmission of ActivateFirmware command) [0] - Automatic (becomes active as the Apply completes, or as download completes if the FD performs an auto-apply)
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, COMMAND_NOT_EXPECTED }

Error completion codes handling:

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

11.10 GetMetaData command format

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

Table 25 – GetMetaData command format

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

Error completion codes handling:

- COMMAND_NOT_EXPECTED: Returned by the UA if this command is received when it is not expected based on the sequence defined to update a firmware component, or if the UA did not previously retrieve the firmware device metadata through the GetDeviceMetaData command.
- INVALID_TRANSFER_HANDLE: Returned from the UA if the transfer handle used in the request is invalid.
- INVALID_TRANSFER_OPERATION_FLAG: Returned from the UA if the transfer operation flag is invalid.

11.11 ActivateFirmware command format

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

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

The FD shall exit from update mode upon the successful completion of this command.

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

Table 26 – ActivateFirmware command format

Type	Request data
bool8	**SelfContainedActivationRequest** True: FD shall activate all self-contained components. False: FD shall not activate any self-contained components.
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE, INVALID_STATE_FOR_COMMAND, INCOMPLETE_UPDATE, ACTIVATION_NOT_REQUIRED, SELF_CONTAINED_ACTIVATION_NOT_PERMITTED }
uint16	**EstimatedTimeForSelfContainedActivation** Amount of time the FD requires to perform a self-contained activation. Measured in seconds after sending this command, the UA should not begin any of the timers listed in Table 2 until after the amount of time present in this field has elapsed. If Self-Contained activation is not requested, this field should be set to zero.

Error completion codes handling:

- INCOMPLETE_UPDATE: Returned by the FD if it is able to determine that not all components are updated completely. The FD will remain in the READY XFER state, and will not perform activation.
- INVALID_STATE_FOR_COMMAND: The FD only expects this command in READY XFER state.
- NOT_IN_UPDATE_MODE: Returned by the FD if it's not in the update mode.
- ACTIVATION_NOT_REQUIRED: Returned by the FD if the new firmware components are already pending activation (such as through a previous ActivateFirmware command), or the activation method was 'automatic' and therefore the component was already activated at the completion of the apply step.
- SELF_CONTAINED_ACTIVATION_NOT_PERMITTED: Retuned by the FD if it does not support Self-Contained activation and the SelfContainedActivationRequest is set to True.

11.12 GetStatus command format

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

Table 27 – GetStatus command format

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

Type	Response data
enum8	**CurrentState** Current state machine state of the FD. 0 – IDLE 1 – LEARN COMPONENTS 2 – READY XFER 3 – DOWNLOAD 4 – VERIFY 5 – APPLY 6 – ACTIVATE
enum8	**PreviousState** The previous different state machine state of the FD. 0 – IDLE 1 – LEARN COMPONENTS 2 – READY XFER 3 – DOWNLOAD 4 – VERIFY 5 – APPLY 6 – ACTIVATE
enum8	**AuxState** Used provide additional information to the UA to describe the current operation state of the FD while in one of the following states (Download, Verify, Apply, or Activate). 0 – Operation in progress. 1 – Operation successful. 2 – Operation failed – FD shall provide Error Code in AuxStateStatus field. 3 – Value used when FD is in IDLE, Learn Components, or Ready Xfer state.
uint8	**AuxStateStatus** 0x00 - AuxState is In Progress or Success. 0x01 - 0x08: Reserved 0x09 - Timeout occurred while performing action. 0x0A - Generic Error has occurred. 0x02 – 0x6F: Reserved 0x70-0xEF - Firmware Device Vendor defined status code. When an FD device uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor define type. For details refer to Table 6. 0xF0 – 0xFF - Reserved
uint8	**ProgressPercent** Used when CurrentState is in the DOWNLOAD, VERIFY or APPLY state. Value range from 0x00 to 0x64 (decimal 0 to 100). This field is optional for an FD. If the FD does not support a progress percent, the value returned shall be 0x65 (decimal 101). If this field is supported by the FD, the value provided in this field represents the percentage complete of the current action (DOWNLOAD, VERIFY, or APPLY). The value is initialized to 0 upon each transition of CurrentState.

Type	Response data
enum8	**ReasonCode** Used when CurrentState is in the IDLE state. Provides the reason for why the CurrentState entered the IDLE state. The value is retained until the next transition to IDLE occurs that will then cause this field to be updated. 0 – Initialization of firmware device has occurred. 1 -- ActivateFirmware command was received. 2 – CancelUpdate command was received. 3 – Timeout occurred when in LEARN COMPONENT state. 4 – Timeout occurred when in READY XFER state. 5 – Timeout occurred when in DOWNLOAD state. 200-255: Firmware Device Vendor defined status code. When an FD device uses a vendor defined status code, it shall also provide Vendor ID information by using either the PCIe or IANA Vendor define type. For details refer to Table 7.
bitfield32	**UpdateOptionFlagsEnabled** 32 bits field used when CurrentState is in the DOWNLOAD, VERIFY, APPLY, or ACTIVATE state, where each non-reserved bit represents an update option that has been enabled by the FD for the transfer of this component image. A ‘1’ in the bit indicates the requested update option flag is enabled. [31:1] – Reserved [0] – Force update of component – FD will perform a force update of the component.

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

11.13 CancelUpdateComponent command format

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

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

Table 28 – CancelUpdateComponent command format

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

Error completion codes handling:

- NOT_IN_UPDATE_MODE: returned by the FD if it’s not currently in update mode.
- BUSY_IN_BACKGROUND: returned by the FD if there is a critical job in the background, and cannot exit from update mode. The UA shall retry after UA_T1.

11.14 CancelUpdate command format

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

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

Table 29 – CancelUpdate command format

Type	**Request data**
--	No request data
Type	**Response data**
enum8	**CompletionCode** value: { PLDM_BASE_CODES, NOT_IN_UPDATE_MODE, BUSY_IN_BACKGROUND }
bool8	**NonFunctioningComponentIndication** True: one or more components will be in a non-functioning state upon the next activation. The non-functioning component bitmap field indicates which components will be non-functioning. False: all components will be functioning. GetFirmwareParameters can be used to determine the individual component version information.
bitfield64	**NonFunctioningComponentBitmap** This field is valid only if the Non-functioning component indication field is set to True. Each bit n corresponds to the nth component passed in the PassComponentTable command. A set bit indicates the component will be in a non-functioning state upon the next activation.

Error completion codes handling:

- NOT_IN_UPDATE_MODE: returned by the FD if it’s not in the update mode.
- BUSY_IN_BACKGROUND: returned by the FD if there are critical tasks already being performed by the device, and cannot exit from update mode. The UA shall retry within UA_T1 interval.

12 Additional information

12.1 Multipart transfers

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

- For initiating a data transfer (or getting the first part of data) using a Get command, the TransferOperationFlag shall be set to GetFirstPart in the request of the Get command.
- For transferring a part other than the first part of data by using a Get command, the TransferOperationFlag shall be set to GetNextPart and the DataTransferHandle shall be set to the NextDataTransferHandle that was obtained in the response of the previous Get command for this data transfer.
- The TransferFlag specified in the response of a Get command has the following meanings:
  - Start, which is the first part of the data transfer
  - Middle, which is neither the first nor the last part of the data transfer
  - End, which is the last part of the data transfer
  - StartAndEnd, which is the first and the last part of the data transfer
- The requester shall consider a data transfer complete when the TransferFlag in the response of a Get command is set to End or StartAndEnd.

Figure 7 shows how the multipart transfers can be performed using the generic mechanism defined in the commands.

In this example, the update agent maintains a copy of the package data provided by the firmware update package. The firmware device gets the package data by using the GetPackageData command. Figure 1 shows the flow of the data transfer.

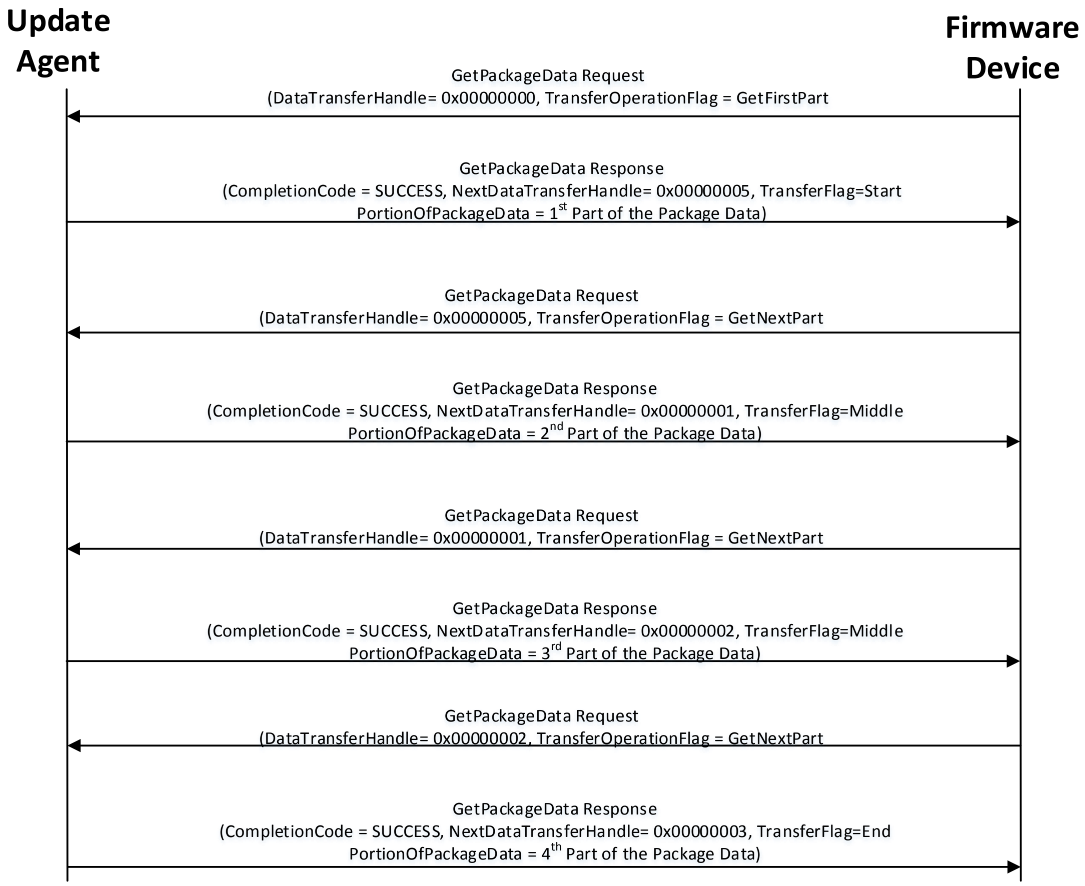
Update Agent
Firmware Device
GetPackageData Request
(DataTransferHandle= 0x00000000, TransferOperationFlag = GetFirstPart
GetPackageData Response
(CompletionCode = SUCCESS, NextDataTransferHandle= 0x00000005, TransferFlag=Start
PortionOfPackageData = 1st Part of the Package Data)
GetPackageData Request
(DataTransferHandle= 0x00000005, TransferOperationFlag = GetNextPart
GetPackageData Response
(CompletionCode = SUCCESS, NextDataTransferHandle= 0x00000001, TransferFlag=Middle
PortionOfPackageData = 2nd Part of the Package Data)
GetPackageData Request
(DataTransferHandle= 0x00000001, TransferOperationFlag = GetNextPart
GetPackageData Response
(CompletionCode = SUCCESS, NextDataTransferHandle= 0x00000002, TransferFlag=Middle
PortionOfPackageData = 3rd Part of the Package Data)
GetPackageData Request
(DataTransferHandle= 0x00000002, TransferOperationFlag = GetNextPart
GetPackageData Response
(CompletionCode = SUCCESS, NextDataTransferHandle= 0x00000003, TransferFlag=End
PortionOfPackageData = 4th Part of the Package Data)

Figure 7 – Multipart package data transfer using the GetPackageData command

12.2 Transport Protocol type supported

PLDM can support bindings over multiple interfaces, refer to DSP0245 for the complete list. This specification requires the transport protocol type to support asynchronous request/response messages that can be sent from either endpoint in order to support the full Firmware Update functionality. All transport protocol types can be supported for the two Inventory commands defined in Table 10.

12.3 Considerations for FD device manufacturers

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

ANNEX A

(informative)

Change log

Version	Date	Description
1.0.0	2016-11-28	
1.0.1	2018-03-28	Updates to UUID field in header, PCI descriptors, and activation state machine transition table

Bibliography

DMTF DSP4004, *DMTF Release Process 2.4,*
http://dmtf.org/sites/default/files/standards/documents/DSP4004_2.4.pdf