OCP GPU FW Update Specification

v1.1

Author (s) :

AMD, Google, Meta, Microsoft, NVIDIA

Contact***: hyperscale-gpu@ocproject.net***

Executive Summary

This document outlines the requirements for firmware updates for GPUs, focusing on secure, efficient, and standardized processes. It includes use cases, terminology, update models, requirements, and Redfish API examples.

Table of Contents

1. Introduction **6**

1.1 Overview **6**

1.2 Assumptions, Constraints,Dependencies 6

1.3 Use Cases 6

1.4 Glossary/Terminology 6

2. Firmware Update Model **9**

2.1 Firmware Update States 10

2.2 Baseboard with Accelerator Management Controller (AMC) 11

2.3 Non-UBB Device with FD 13

2.4 Firmware Device State Machine Diagram 16

3. Firmware Update Requirements **19**

3.1 Firmware Update 19

3.1.1 Secure firmware update 19

3.1.2 Firmware Copy time 19

3.1.3 Firmware activation 19

3.1.4 Firmware Update protocols 20

3.1.5 Use of Force Update 20

3.1.6 Firmware update dependencies 21

3.1.7 Firmware date compatibility 21

3.1.8 Non-disruptive updates 21

3.1.9 Firmware update synchronization 21

3.1.10 Restore previous image upon cancellation of firmware update 21

3.2 Firmware Rollback Prevention 21

3.2.1 Rollback prevention model 22

3.2.2 Query/Disable/Enable Irreversible Configuration 22

3.2.3 Query Firmware Component Identities 24

3.2.4 Query Firmware Security Version Number 25

3.2.5 Update Minimum Security Version Number 27

3.2.6 Firmware update package field for SVN 29

3.3 Firmware Image Format 29

3.4 Firmware Image Size 30

3.5 Firmware Interchangeability 30

3.6 Single shot firmware update 31

3.7 Firmware Update Failure 31
3.8 Firmware Image redundancy 33
3.9 Firmware resiliency 34
3.9.1 Single Failure 34
3.9.2 Double Failure 34
3.10 Firmware telemetry 34
3.11 Debug messages for runtime DC Ops 34
4. Redfish Update Service 35
4.1 Common properties 35
4.2 Update Service and Software Inventory 36
4.2.1 UpdateService 36
4.2.2 Software Inventory 40
4.3 TaskService 43
4.3.1 Task 43
4.4 Redfish Error Reporting for Firmware Update 45
4.4.1 Message Registry 45
4.4.2 Immediate Failures 47
4.4.3 Failure during Asynchronous Operation 48
4.4.4 Failure during activation/background-copy operation 50
4.5 Redfish API Examples for Multipart Push Update and Split Copy/Arm support 51
5. PLDM Update 56
5.1. PLDM Update Commands 56
5.2 PLDM Update Timing Requirements 58
6. File Format 59
6.1 File Format 59
6.2 Package Security 60
7. References 62
8. License 62
About Open Compute Foundation 63
Appendix A. [Redfish API Example] 64
Appendix B 75

1. Introduction

1.1 Overview

This document specifies the GPU firmware(FW) update requirements and Redfish/PLDM interfaces to perform FW updates on GPU products. This document covers the FW Update model for discrete GPUs and Universal Base-board based GPU systems containing management controllers. This document further captures the e2e flows from a Data Center orchestrator perspective.

1.2 Assumptions, Constraints,Dependencies

- Hyperscale partners own Host BMC.
- Requirements are for vendor HW board and/or firmware devices.
- Host BMC implementation of these requirements will be co-developed by the Hyperscaler and the Vendor.

1.3 Use Cases

The following table lists the use cases.

Use Case	Description
Universal base boards	Vendor provides a proprietary baseboard, with capabilities for power sequencing, firmware update, telemetry, and Accelerator Management Controller. Example Nvidia HGX cards
Discrete Accelerator Device	Vendor provides standard PCIe cards, without accelerator BMC. Example Nvidia PCIe cards (GPU and/or NIC)

Table 1: use cases

1.4 Glossary/Terminology

Activation	This is the operation to load and run the FW in a device. This usually involves a reset or a device specific mechanism.
Arm	This is the operation to mark the FW image as the active image for the device to load on next use. The device can be forced to load and use the active image via a device reset or some other device specific mechanism.
Copy	This is the operation of sending a FW Image or Bundle to a GPU device or subsystem. The expectation is that this does not disrupt the normal operation of the device during this process. The end state of this operation is the successful writing/storing/flashing of the FW Image within the GPU device subsystem.
Flashing	Storage of a FW Image in a non-volatile media typically attached to a device/RoT and used by the device to load its FW for operation. This does not imply any particular type of media or partition.
Interchangeability	The property of a product whose individual units are interchangeable despite implementation differences. Interchangeable units must adhere to the same product definition in terms of form, fit, and compute value.
Firmware Interchangeability	The ability for a single product firmware image or firmware bundle to support interchangeable units.
FW Bundle	one or more firmware payload packaged per an aligned specification
Orchestrator	Initiator of FW updates, which controls the entire flow of transfering, updating and activating the firmware for one or multiple nodes.
PRoT	Platform Root of Trust. Entity in the platform that measures each component in the platform.
precheck	aspect of Cloud Orchestrator verifying the device firmware versions, security versions and comparing with new incoming versions
postcheck	The aspect of Cloud Orchestrator verifying if the device has accepted the

	firmware update and making use of the new firmware
RoT	Root Of Trust. This could be a device level RoT or Platform entity.
Update	Update is the entire process of copying, arming and activating a specific FW to run on a device.

Table 1: Firmware Update Terminology

Term	Elaboration	Description
AP	Application Processor	Examples of APs include GPU, FPGA, and PCIE Switch.
AP_FW	Application processor firmware	Firmware that is associated with an AP.
AMC	Accelerator Management Controller	A controller that acts as the management entity for a UBB based GPU system. It typically exports Redfish but may export a custom interface as well.
CS	Chip Select	Chip select pins are often used with the serial peripheral interface (SPI) protocol to select one of multiple SPI devices that share the same data lines.
CSP	Cloud Service Provider	
EC_FW	Embedded Controller Firmware that runs on the External root-of-trust	
ERoT	External Root of Trust (RoT)	A RoT that is external to a device. To provide flash security, this controller can sit between a

Term	Elaboration	Description
		device and its flash storage.
FLR	Function level reset	Function level reset as defined by PCIE specifications.
FD	Firmware Device	A hardware device running microcode or embedded SW.
Host BMC	Host Baseboard management controller	A microcontroller that is responsible for the overall management of a system. It contains many components, which include monitoring and control of power sequencing, thermal elements, device telemetry, and firmware updates/recovery.
InB	In-Band	Refers to the data plane of a device that is in line with the host operating system under which a device is operational.
MCTP	Management Control Transport Protocol	Refers to the DTMF MCTP standard (DSP0236 and related specification).
OOB	Out-of-band	Refers to the management plane of a device that is not in line with the host operating system under which a device is operational.
PEC	Packet Error Code	In the SMBus spec, it is CRC-8 calculated over all message bytes.
PEXRST		PCIE Reset as defined by the PCIE specifications.
PLDM	Platform Level Data Model	Refers to the DMTF PLDM standard (DSP0248, DSP0267,

Term	Elaboration	Description
		and related specification).
SBR	Secondary bus reset	Secondary bus reset as defined by the PCIE specifications.
UA	Update Agent	Update Agent as defined in the PLDM for Firmware Update specification (DMTF DSP0267).

Table 2: Firmware Device Terminology

2. Firmware Update Model

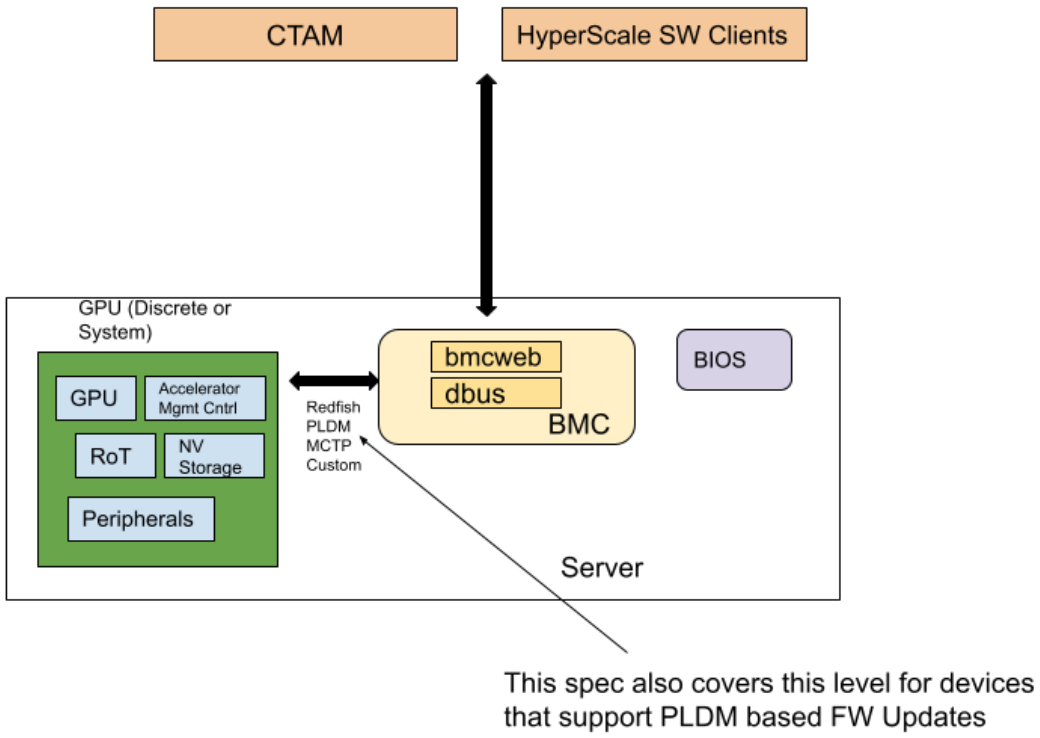


Figure 1: Hyperscaler Firmware update model

This specification assumes a device and management model shown in Figure 1. There are two classes of GPU products being modeled, a discrete GPU such as a PCIe card and an UBB or GPU system that contains one or more GPUs and other components (like PCIe switches, fabric elements etc.). The management model assumes a BMC connected to the GPU product to manage it and also provides an interface to the Hyperscale orchestrator (and other clients). The northbound interface from the BMC is based on Redfish. For UBB based GPU system, the model assumes a management controller provides a Redfish based interface to the BMC. This will be referred to as the "Accelerator Management Controller". For GPUs that don't expose Redfish (like many discrete PCIe cards), the management protocol between the BMC and the GPU is expected to be PLDM over MCTP.

2.1 Firmware Update States

Distributed computing systems leverage multiple networked Server Nodes with thousands of hardware accelerators, such as GPUs and TPUs to process huge amounts of data and achieve high performance and efficiency. Hyper scalars who operate these distributed computers usually perform gang firmware updates on these nodes . One of the strong desires is to use the common firmware version across all the Nodes of the distributed computing nodes to achieve common TCO/security objectives and to reduce the overall downtime for their customers due to Firmware Updates. Some of the Modern GPU design firmware update time is 15 mins firmware staging time and several minutes to activate the new firmware across reset. In the current PLDM/Redfish definition, FW Staging and Activation needs to happen back-to-back on a relatively short time scale. Thus, the Customer downtime is for the entire duration of FW staging and Activation.

As a solution to reduce this downtime, Hyper scalars could opt-in to stage the firmware while the Node is in operation and perform gang activation in future time. But this method is susceptible to selected Nodes getting activated with newer firmware due to accidental restarts like system power loss, restart on system crash, device resets to address VM teardown etc. The concept of explicit arming of the firmware is introduced to avoid system/device accidentally consuming the newly staged firmware.

The FW Update process consists of various steps and phases. The basic steps of a FW Update operation from a Hyperscaler Orchestrator/BMC point-of-view is:

- FW Copy: This copies the FW package to the GPU FW device.
- FW Arm: This marks the FW to be ready for activation

- FW Activate: This loads and executes the FW on the device. This is usually accomplished by a device reset or power cycle.

The following sections describe the theory of operation of a reference FW Update process. This provides a model for the requirements described in Section 3.
The next subsections describe a reference firmware update sequence for two scenarios,

- a complex baseboard with a Accelerator Management Controller (AMC)
- a simpler board such as a PCIe card which implements Firmware Device (FD)

2.2 Baseboard with Accelerator Management Controller (AMC)

The figure below shows AMC updating firmware of all devices on a vendor baseboard on behalf of the host BMC.

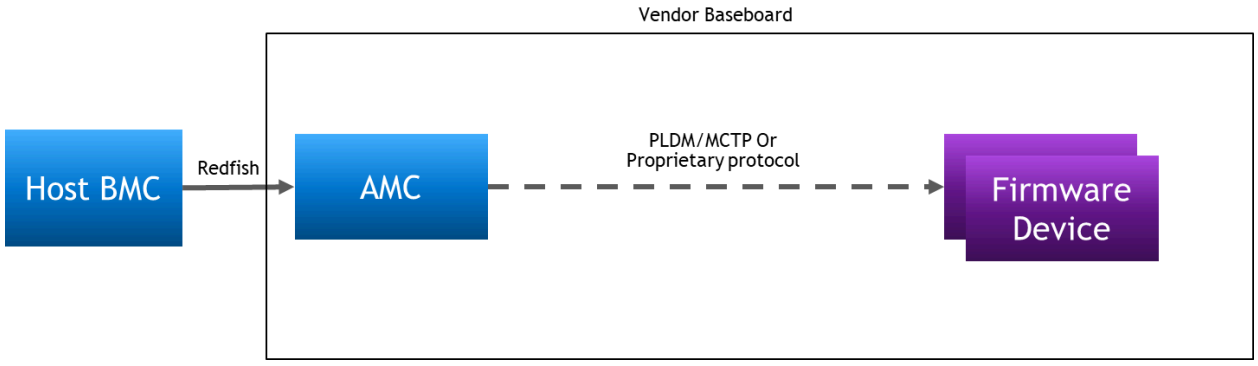


Figure 2: GPU Firmware device with AMC

Host BMC uses Redfish Firmware Update (by using the UpdateService schema) to send to AMC firmware updates of all devices in the baseboard packaged in a PLDM bundle (as per DMTF DSP0267). AMC then downloads images to all the devices on the baseboard, e.g using PLDM or a proprietary protocol.

The sequence diagram below shows the end to end update process from the CSP's Orchestrator to the Firmware Device. The diagram uses PLDM protocol to illustrate the interaction between the AMC and Firmware Device but any equivalent proprietary protocols that achieve the same functionality are acceptable. Update happens in three steps:

- Pre-Check, where the Orchestrator checks the firmware versions and firmware health of the devices.
- Update, which happens in two steps:

    - The images are copied, verified and then applied on the firmware devices during production without any service impact, and then,
    - The images are activated at an appropriate time (e.g. during the next activation cycle like a maintenance window) after the Orchestrator has issued an explicit *Arm* message to make the FW ready to be activated in the next activation cycle.
- Post-Check, where the Orchestrator confirms the update is successful by checking the firmware versions and the health of the firmware devices.

Here are the steps for firmware update:

1. The AMC, on boot, discovers the update capable PLDM firmware devices. The Orchestrator detects the AMC and recognizes it as an updatable device.
2. The Orchestrator checks the firmware versions and the health of the devices on the baseboard from the AMC (via the host BMC)
3. The Orchestrator then uses Redfish UpdateService to copy the firmware bundle to the AMC.
4. Upon successful receipt of the firmware bundle, the AMC sends the Upload complete message to the Orchestrator. The Orchestrator then periodically queries the AMC with Redfish Task Progress until the images are applied to the firmware devices.
5. Upon receipt of the firmware bundle, the AMC identifies the right firmware for each device in the baseboard and copies the firmware to all the devices.
6. Devices apply/save the images on their non-active non-volatile memory after verifying the images. Verification may include authenticating the signatures on the images and any security rollback restrictions.
7. If the verification fails, the firmware will not be activated, and the non-active area will be restored with the image from the active region.
8. Once images are applied/saved in the firmware devices, the AMC notifies the Orchestrator (via the host BMC).
9. The Orchestrator sends the measurements of the new firmware to the PROT so it can start using them for attestation.
10. The Orchestrator then sends an *Arm* command to the AMC.
11. AMC then relays the command to all the firmware devices, which then *Arm* their new images.
12. Once the Orchestrator reboots/power cycle/resets the system, the new firmware on the devices are activated.
13. The Orchestrator does a post check of the firmware versions and the health of the devices to make sure the update was successful.

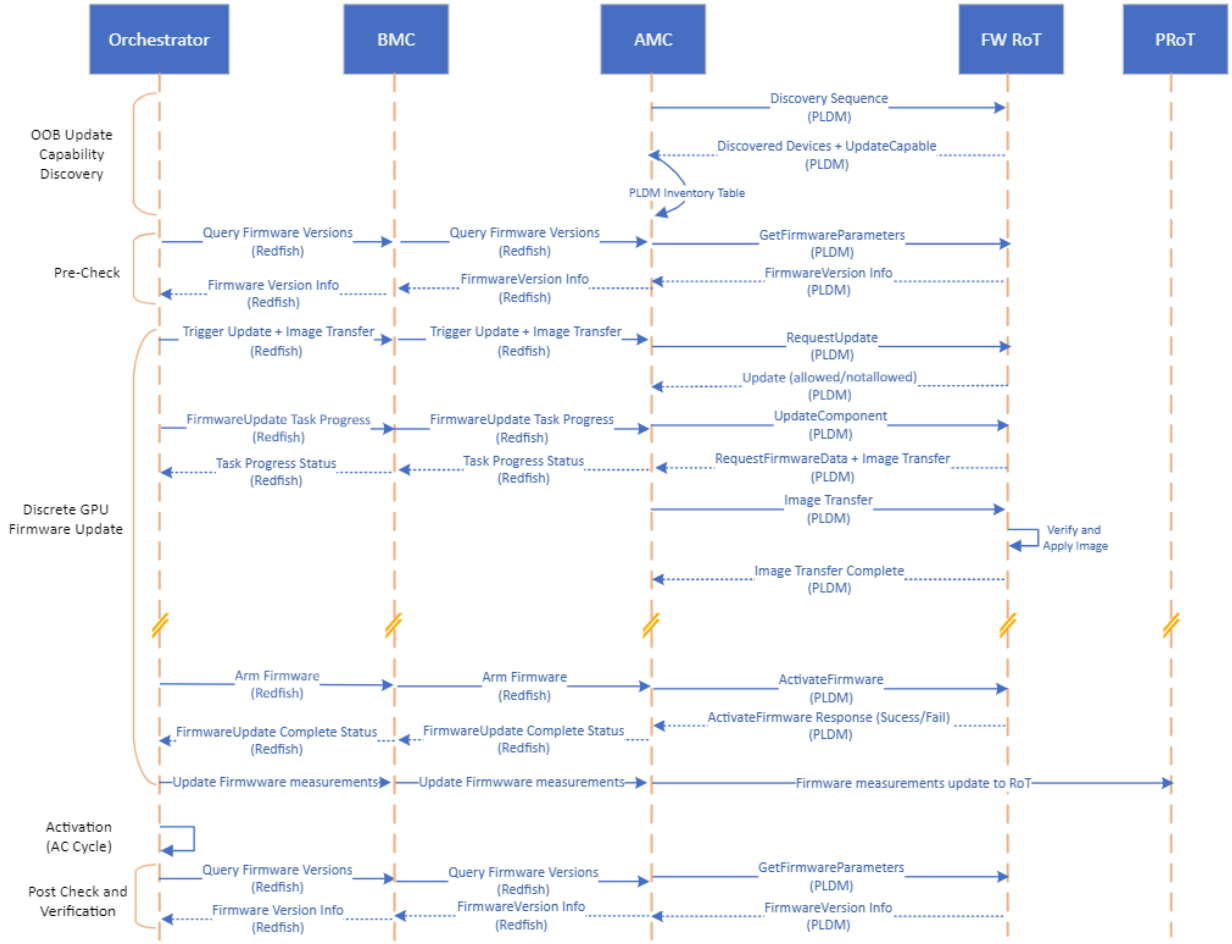


Figure 3: Firmware update sequence diagram with AMC

2.3 Non-UBB Device with FD

The figure below shows the host BMC directly updating firmware of all devices on a vendor board, when a vendor board doesn't have an AMC. Host BMC uses PLDM over MCTP to copy images to all the devices on the board.

Figure 4: GPU Firmware device without AMC

The sequence diagram below shows the end to end update process from the CSP's Orchestrator to the Firmware Device. Update happens in three steps:

- Pre-Check, where the Orchestrator checks the firmware versions and firmware health of the devices.
- Update, which happens in two steps:
    - The images are copied, verified and then applied on the firmware devices during production, and then,
    - The images are activated only during a maintenance window after the Orchestrator has issued an explicit *Arm* message.
- Post-Check, where the Orchestrator confirms the update is successful by checking the firmware versions and the health of the firmware devices.

Here are the steps for firmware update:

1. The Orchestrator checks the firmware versions and the health of the devices on the baseboard from the host BMC.
2. The Orchestrator then uses Redfish UpdateService to download the firmware bundle to the host BMC.
3. Upon successful receipt of the firmware bundle, the host BMC sends the Upload complete message to the Orchestrator. The Orchestrator then periodically queries the host BMC with Redfish Task Progress until the images are applied to the firmware devices.
4. Upon receipt of the firmware bundle, the host BMC identifies the right firmware for each device in the baseboard and copies the firmware to all the devices.
5. Devices apply/save the images on their non-active non-volatile memory after verifying the images. Verification may include authenticating the signatures on the images and any security rollback restrictions.
6. If the verification fails, the firmware will not be activated, and the non-active area will be restored with the image from the active region.
7. Once images are applied/saved in the firmware devices, the host BMC notifies the Orchestrator.
8. The Orchestrator sends the measurements of the new firmware to the PROT so it can start using them for attestation.
9. The Orchestrator then sends an *Arm* command to the host BMC.

10. The host BMC then relays the command to all the firmware devices, which then *Arm* their new images.
11. Once the Orchestrator reboots/powercycle/resets the system, the new firmware on the devices are activated.
12. The Orchestrator does a post check of the firmware versions and the health of the devices to make sure the update was successful.

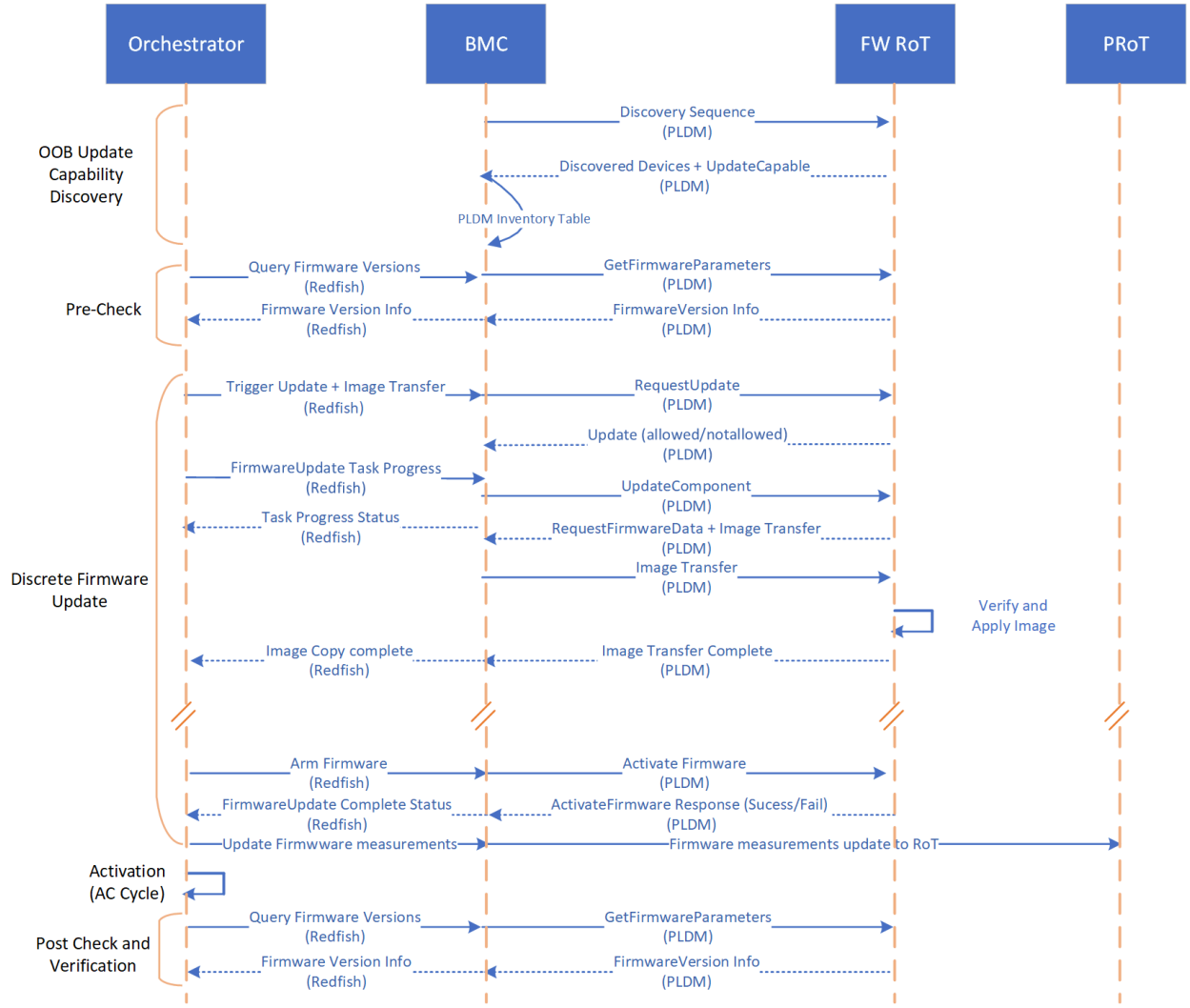


Figure 5: Firmware update sequence diagram without AMC

2.4 Firmware Device State Machine Diagram

The figure below shows the state machine diagram for a firmware device during firmware update

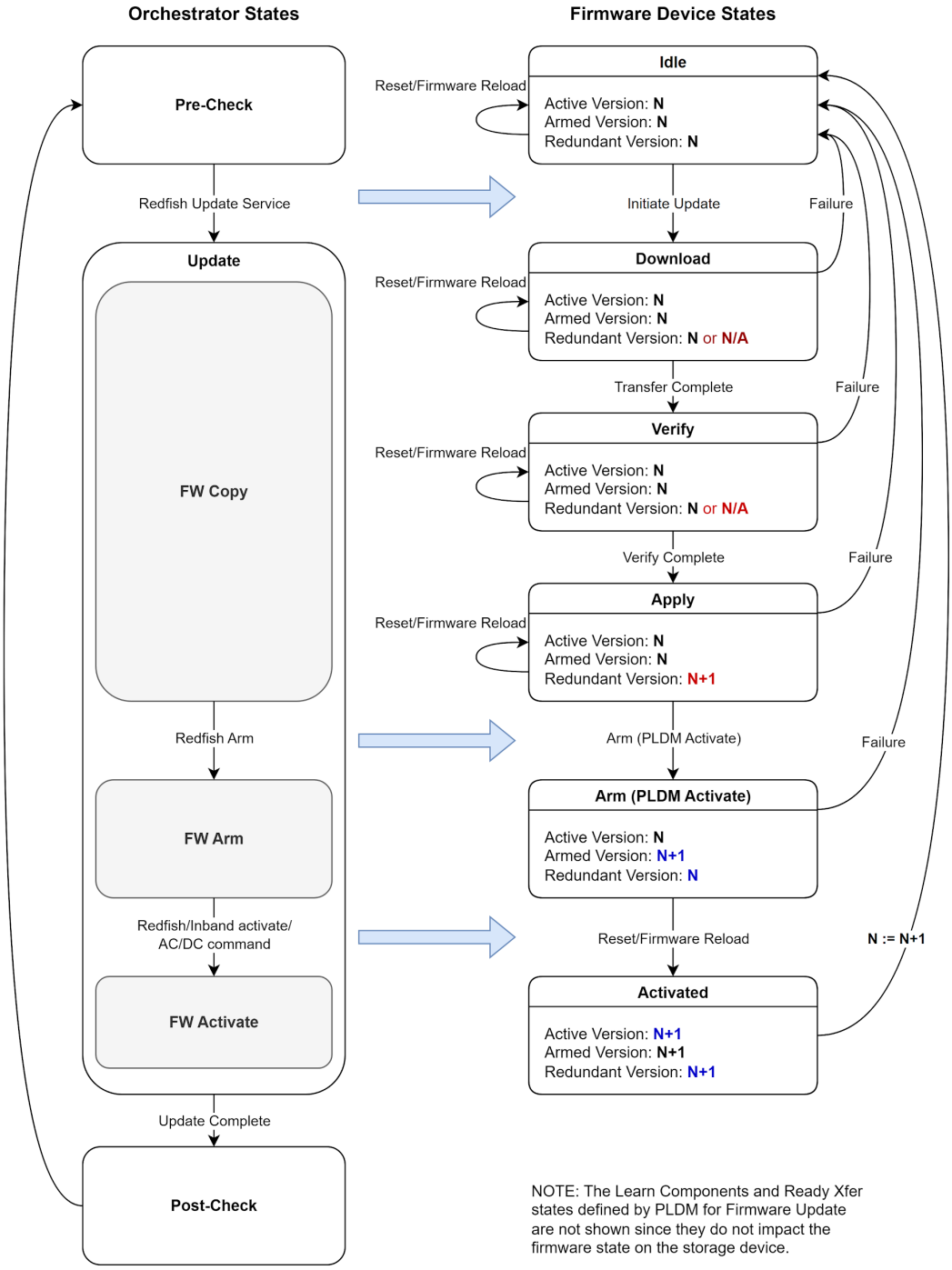


Figure 6: Firmware update device states

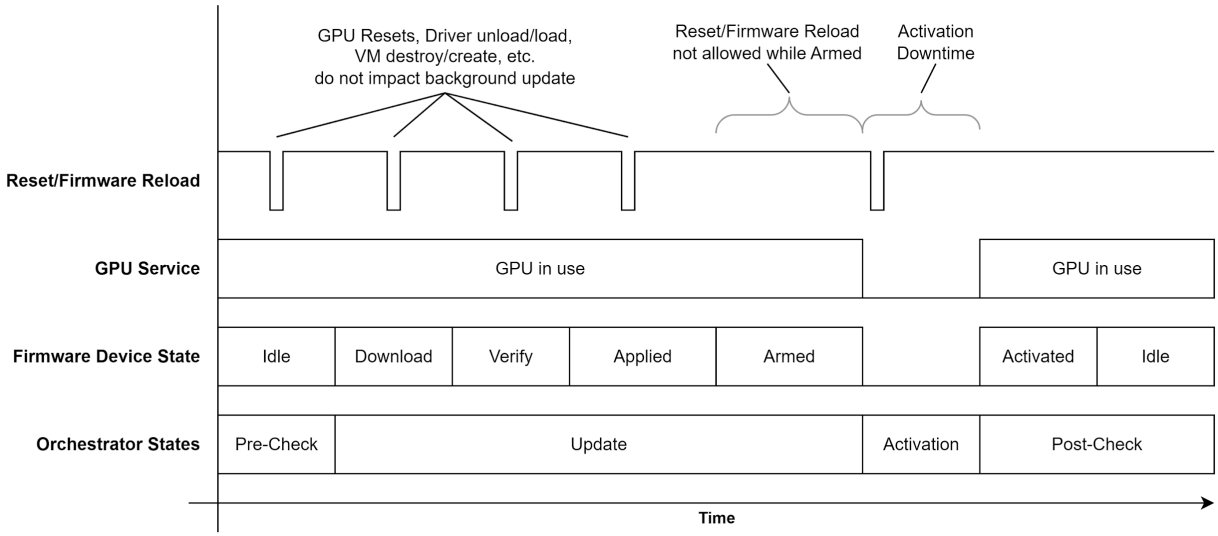


Figure 7: Firmware Device background update

State definition table

State Name	Description
Idle	Default state, Firmware Device (FD) is fully initialized and ready to receive the firmware update/firmware inventory related commands
Download	After receiving the appropriate Redfish command to update a firmware component, the Update Agent (UA) verifies the PLDM package checksum and unwraps the component images. Upon finding an FD and component(s) matching a device and component from the PLDM package, the UA initiates a firmware update with the FD. The FD moves to the download state and communicates with the UA to transfer the component image(s).

Verify	In this state the FD (or the Embedded ROT) performs security verification (signature/HASH) of the firmware images, verification algorithm used is GPU specific but recommended once are >= SHA 256/RSA 2048.
Apply	Upon completion of download and successful verification of the component image, if the image was temporarily stored in a staging area, the FD copies the firmware image into a bootable location in nonvolatile storage. If the image was downloaded directly into a redundant bootable location in nonvolatile storage, then no additional copy is required. In this state, the FD may make additional preparations to boot the new image, but must not arm the image for activation.
Arm	Upon successful image application, Hyper scalar Update Agent should arm the image for activation. For devices that support activation by AC power cycle, DC power cycle, or system reboot, the PLDM Activate Firmware command is sent to arm the firmware for activation. For devices that support self-contained activation, the PLDM Activate Firmware command must be deferred to the update activation stage.
Activate	Upon successful arming of firmware images, Hyper scalar update agent requests FD to perform image activation. For devices that support activation by AC power cycle, DC power cycle, or system reboot, the appropriate reboot or power cycle method is executed. For devices that support self-contained activation, the PLDM Activate Firmware command may be issued to the FD. Triggered by the activation method, the FD consumes the new firmware. Once the firmware has been loaded, the FD returns to the IDLE state.

Table 3: Firmware update State definition

Initial State	Next State	State Trigger	On Error
Idle	Download	Redfish Update command	Retry download

Download	Verify	Redfish Task monitor	Retry download
Verify	Apply	implicit	Retry download
Apply	Arm	Redfish Arm command	Retry download
Arm	Activate	Redfish/inband activate /AC/DC command	Retry download

Table 4: Firmware update State transition table

3. Firmware Update Requirements

3.1 Firmware Update

The product shall support OOB secure firmware update of all its mutable components. The update must be designed to be secure, fast, efficient, standards based, and reduce system downtime to complete the update.

3.1.1 Secure firmware update

Vendors must provide secure updates of firmware. Secure update may be implemented by using ERoTs or iRoT's with all the firmware devices. The ERoT/iRoT performs the following functions:

- Verifies the cryptographic signatures on every firmware update.
- Enforces a rollback policy to prevent firmware downgrade attacks.
- Enforces key revocation checks to ensure that a compromised key cannot be used to sign firmware updates.

Vendors must also provide mechanisms to write-protect firmware updates from the in-band path i.e. prevent FW update by any inband agents (host CPU). Write protection must be configurable, e.g via EROT to block updating firmware flash. In addition, a hardware based write protect, such as user controlled GPIO shall be available to disable/override programmable configuration. Write protect should be disabled by default to avoid firmware update lockout.

3.1.2 Firmware Copy time

Firmware updates for all devices shall be performed as described in Section 2, first requiring downloading/copying images to the devices and then activating them on receiving explicit

command. Vendors must design firmware updates to keep copying of the image to the devices to a minimum and must not cause any disruption in services. Copy of firmware images to devices should be in the order of minutes for baseboards with Accelerator Management Controllers and order of seconds for firmware devices for boards without Accelerator Management Controllers. To enable fast copy of images to the devices, vendors may use fast physical transport such as USB, SPI, I3C.

3.1.3 Firmware activation

After devices have copied the images, they will activate the image only upon receiving explicit command. The devices may not use auto self-contained activation (as defined in DMTF DSP0267).

The following activation methods may be required:

- AC Power Cycle
- DC Power Cycle
- Warm Reboot (PEXRST, FLR/SBR, and Software)
    - Refer to PCIE specifications for how to reset PCIE devices through PEXRST and FLR/SBR.
    - The term *Software* refers to the software methods of rebooting/restarting/resetting a component by using command line tools or the Redfish APIs to trigger the resets.

Activating images may require system downtime and may disrupt service. If activation is disruptive, down time shall be in the order of minutes for baseboards with Accelerator Mgmt Controller and order of seconds for baseboards without Accelerator Mgmt Controller.

3.1.4 Firmware Update protocols

For baseboards containing accelerator BMC, host BMC will use Redfish Firmware Update (by using the UpdateService schema) to send accelerator BMC firmware updates of all devices in the baseboard packaged in a PLDM bundle (as per DMTF DSP0267). Accelerator BMC will then use PLDM to download images to all the devices on the baseboard.

For boards without accelerator BMC, host BMC uses PLDM over MCTP (as per DMTF DSP0267) to download images to all the devices on the board.

Using these standards for firmware updates helps with interoperability, reduces or eliminates the need for interdependent device firmware, and essentially makes it vendor-neutral.

3.1.5 Use of Force Update

Vendors shall support the option to force FW update for the following two cases:

1. Idempotency - By default, devices shall perform FW version check and skip update if version matches to the one from the package. However, if the Force Update flag is configured in the Redfish Update Service then the FW shall be overwritten.

2. Downgrades - If the FW version is older than the version on the device, then by default, the device will not be downgraded. However, if the Force Update flag is set then the device will be downgraded modulo and security-based rollback constraints.

Force Update flag must be used in the POST command in the Update Parameter setting of Redfish Update Service. The bit in the PLDM bundle header that sets the force bit (Component Image Information section, UpdateOptionFlags bitfield 32 reserved[0]) must be a don't care and ignored by the updater

3.1.6 Firmware update dependencies

Ideally there shall be no dependency between FW of different devices in a product. This is applicable for production firmware releases. Furthermore, one must be able to directly update between any two FW versions without any update to an intermediate version, as long as there are no rollback restrictions as defined in Section 3.2. At a minimum, the supported and tested FW update path shall include:

1) Factory shipped version to latest released version
2) Update from FW version released 6 months prior to the latest version.
3) Update from a newer version of FW to an older version of FW released 6 months prior

3.1.7 Firmware date compatibility

Configuration and meta data for the firmware data stored in non-volatile memory will be backward and forward compatible with other versions of the firmware. This is necessary to enable updates between different versions without losing stored data, such as configs, tuning parameters etc.

3.1.8 Non-disruptive updates

As firmware update copy may happen during production, it shall not disrupt any production activity. Specifically, the following operations shall not be impacted and continue to meet any SLAs:

- Runtime stack and application performance
- Inband telemetry
- Out of band telemetry
- Security operations such as Attestation.

3.1.9 Firmware update synchronization

Firmware devices shall prevent firmware update from starting if the previous firmware update is still in progress.

3.1.10 Restore previous image upon cancellation of firmware update

Firmware Device shall provide capability to restore to previous firmware if a firmware update operation is interrupted or canceled for any reason.

3.2 Firmware Rollback Prevention

Firmware should not be rolled back to older versions which may have security flaws. This is commonly referred to as Rollback Prevention, Rollback Protection, or Anti-Rollback. The following sections address requirements to achieve this capability.

3.2.1 Rollback prevention model

Each FW binary image shall have a 'FW Security Version' (SVN) attribute. There shall be two copies of the SVN. The FW binary image itself shall have a copy in the signed FW binary image for consumption by the Root of Trust for Update (RTU). The PLDM firmware update package shall contain a copy of the SVN for use by the update orchestrator or UA.

The device SOC shall maintain a representation of 'Minimum FW Security Version' (MIN_SVN) per component in EFUSEs. Multiple physical instances of a component that are represented by a single component (as defined by PLDM for Firmware Update) may use a single set of EFUSEs for MIN_SVN.

Upon firmware update, the RTU will derive the MIN_SVN from the EFUSEs and compare it to the SVN field in the signed FW binary image. If the FW image SVN is not equal to or greater than the MIN_SVN, the device must indicate relevant error and stop the update. Likewise, upon boot, the root of trust shall also compare the FW image SVN to the MIN_SVN derived from EFUSEs. If the FW image SVN is not equal to or greater than the MIN_SVN, the device must indicate an error and stop further progression of boot.

The device shall support an explicit command to update the MIN_SVN stored in EFUSEs. The hypersaler chooses when to update the MIN_SVN; the security version should not be updated automatically during firmware update. The Hyperscaler chooses a MIN_SVN less than or equal to the FW image SVN based on assessment of the security flaws and the requirements for their fleet.

The device shall support commands for reporting the FW binary image(s) SVN value(s) and the MIN_SVN derived from EFUSEs. The reported SVN may be used by the Hyperscaler to determine the permissible values for updating MIN_SVN. The reported MIN_SVN may be used by the update orchestrator or UA as part of update precheck, similar to comparing the Component Comparison Stamp from the firmware update package to the Component Comparison Stamp reported in Get Firmware Parameters.

3.2.2 Query/Disable/Enable Irreversible Configuration

This specification requires commands that result in irreversible configuration changes, such as commands that program OTP or EFUSE. The commands to make these changes are not authenticated commands, are not restricted by the PLDM FW update state machine, and may be prone to inadvertent use during development and fuzz testing. An “enable” command is required to limit the availability exposure of such commands.

This command is used to enable commands that result in irreversible configuration changes, such as commands that program OTP or EFUSE. The enable state is cleared by any of the following events or actions:

- Issuing this command with RequestType = 1 (disable)
- Issuing a command that requests an irreversible configuration change.
    - The enable state is cleared regardless of success or failure of the command
    - The enable state is cleared even if the request is effectively a no-op
- Timeout of 120 seconds without one of the above events
- Reset of the device

This command template describes the requirements for the command to query, disable, and enable irreversible configuration.

Request data		
Field	**Type**	**Description**
Request type	enum8	Select response or command. The following values are valid: 0 – Query irreversible configuration state 1 – Disable irreversible configuration changes 2 – Enable irreversible configuration changes
Response data: Request type = 0, Total response length 1 byte		
Field	**Type**	**Description**
Irreversible Configuration State	enum8	Possible values: 0 – Irreversible configuration changes disabled 1 – Irreversible configuration changes enabled
Response data: Request type = 2, Total response length 8 bytes		
Field	**Type**	**Description**
Nonce	uint64	Nonce to be supplied to subsequent command that results in irreversible configuration change.

Table 5: Query/Disable/Enable Irreversible Command requirements

3.2.3 Query Firmware Component Identities

Devices shall support capability to query valid firmware components identities. Commands that operate on a firmware component use three request parameters to identify the component: Component Classification, Component Identifier, and Component Classification Index. The definitions of these parameters are consistent with the PLDM type 5 specification's use to identify firmware components. Although PLDM type 5 GetFirmwareParameters may be used to discover firmware component identities, an OCP MCTP Vendor Defined Message (VDM) is defined to discover the component identities to allow these commands to be used independently of PLDM type 5.

This command template describes the requirements for the command to query firmware component identities.

Request data		
No request data		
Response data		
Field	**Type**	**Description**
Component Count	uint8	The number of valid component identities
Component Classification	uint16	Vendor specific component classification information. Refer to PLDM type 5 specification (DSP0267) for specific values.
Component Identifier	uint16	FD vendor selected unique value to distinguish between component images
Component Classification Index	uint8	Used to distinguish identical components that have the same classification and identifier which can use the same component image but the images are stored in different locations in the FD.

Table 6: Query Firmware Component Identities Command requirements

3.2.4 Query Firmware Security Version Number

Devices shall support capability to query SVN from installed FW images and MIN_SVN from fuses. The device shall support querying the SVN from the active FW image and any redundant FW images. The set of SVN values reported must be the same set used to enforce a maximum MIN_SVN value when a MIN_SVN update is requested. The device shall support querying the MIN_SVN from EFUSE and the pending MIN_SVN value to be written to EFUSE upon completion of the method indicated in response to Update Minimum Security Version Number.

This command template describes the requirements for the command to query the SVN value of installed FW images and MIN_SVN in EFUSE.

Request data		
Field	**Type**	**Description**
Component Classification	uint16	Vendor specific component classification information. Refer to PLDM type 5 specification (DSP0267) for specific values.
Component Identifier	uint16	FD vendor selected unique value to distinguish between component images
Component Classification Index	uint8	Used to distinguish identical components that have the same classification and identifier which can use the same component image but the images are stored in different locations in the FD.
Response data		
Field	**Type**	**Description**
Active Component Security Version Number	uint16	SVN of the active firmware image.
Pending Component Security Version Number	uint16	SVN of the armed firmware image that will become active on next activation of the component. If there is no pending firmware image, then this field is used to report the SVN of a redundant bootable firmware image. If no pending image and no redundant firmware image, then this should be set to 0.
Minimum Security Version Number	uint16	MIN_SVN stored in EFUSE.
Pending Minimum Security Version Number	uint16	MIN_SVN value to be written to EFUSE upon completion of the method indicated in response to Update Minimum Security Version Number. If no MIN_SVN update is pending, set to 0.

Table 7: Query SVN Command requirements

3.2.5 Update Minimum Security Version Number

Devices shall support capability to update the MIN_SVN stored in their EFUSEs upon receiving an explicit command. Updating the security version should not be updated automatically during firmware update. Further, updates of the security version should not cause a disruption to the service.

An OCP MCTP Vendor Defined Message (VDM) is defined for updating the MIN_SVN stored in EFUSEs.

The requested MIN_SVN value must be less than or equal to the least of the active firmware SVN and any redundant firmware images' SVNs; updating MIN_SVN must not undermine firmware redundancy. When requesting the maximum permitted value, the maximum allowed value determined from the SVN of the installed images will be stored in EFUSEs.

A device that supports immediate update of EFUSE shall validate the requested MIN_SVN value at the time of the request. If the requested MIN_SVN is valid, then the device shall immediately attempt to update EFUSE. The response shall indicate if the update was attempted, whether the EFUSE update was successful, and whether any additional action (i.e. reset) is required to complete the update.

A device that requires an additional action, such as reset, to update EFUSE shall validate the requested MIN_SVN value at the time of the request. If the requested MIN_SVN is valid, then the device shall store the request in non-volatile memory to be processed upon the required additional action. If the device is power cycled, the EFUSE update shall still be executed when triggered by the required action, typically a device reset. After the update command is accepted and before the EFUSE change is executed, the Query Firmware Security Version Number response shall include the pending MIN_SVN.

While a MIN_SVN update is pending, firmware update is not allowed. A firmware update request shall return an error indicating that a MIN_SVN EFUSE update is pending. A subsequent request can change the pending value or "cancel" the update by requesting a MIN_SVN equal to the current MIN_SVN stored in EFUSE. Upon canceling the update, the prescribed reset action indicated in the response is still required to re-enable firmware update.

After reset and before executing the EFUSE update, the pending update value must be validated again. If the requested update value is invalid or if the EFUSE update otherwise fails, an error

shall be recorded. A failure is otherwise indicated by the Query Firmware Security Version Number response indicating there is no pending MIN_SVN and the current MIN_SVN in EFUSE remains unchanged.

Updating MIN_SVN is not permitted while a firmware update is pending. The firmware update must complete activation before an update MIN_SVN request is permitted. If background copy is enabled, then background copy must also complete before an update MIN_SVN request is permitted. A typical MIN_SVN update sequence follows these steps:

1. Perform PLDM type 5 firmware update, including activate.
2. Complete firmware activation by issuing reset or power cycle as required
3. If background copy enabled, complete background copy
4. Request nonce using Enable Irreversible Configuration command
5. Request MIN_SVN update
6. Take additional action (i.e. reset) as indicated in response

This command template describes the requirements for the command to update the MIN_SVN value in EFUSE.

Request data		
Field	**Type**	**Description**
Request type	enum8	Select response or command. The following values are valid: 0 – Update MIN_SVN to most restrictive permitted value 1 – Update MIN_SVN to specified value
Component Classification	uint16	Vendor specific component classification information. Refer to PLDM type 5 specification (DSP0267) for specific values.
Component Identifier	uint16	FD vendor selected unique value to distinguish between component images
Component Classification Index	uint8	Used to distinguish identical components that have the same classification and identifier which can use the same component image but the images are stored in different locations in the FD.
Nonce	uint64	Nonce obtained from Enable Irreversible Configuration command.

Requested Minimum Security Version Number	uint16	MIN_SVN requested to be stored in EFUSE. For Request Type = 0: Ignored
Response data		
Field	**Type**	**Description**
Update Methods	enum32	Method required to complete the EFUSE update: [31:19] – Reserved. [18] – Function Level Reset (FLR) [17] – Hot Reset (SBR, link disable) [16] – Warm Reset (PERST#) [15:6] – Reserved. [5] – AC power cycle [4] – DC power cycle [3] – System reboot [2] – Medium-specific reset [1] – Reserved. [0] – Automatic. EFUSE updated at completion of request.

Table 8: Min SVN Update Command Requirements

3.2.6 Firmware update package field for SVN

The firmware update package specification does not provide a method for per-component OEM extensions, unlike that done with MCTP VDM to extend the device API. The firmware update package would benefit from having the SVN in the component image headers for each FW binary. This copy of the SVN would be used by consumers of the firmware update package, such as the update orchestrator or the UA, in a similar way to how component comparison stamp can be used to make decisions as to whether to proceed with firmware update.

OCP will work with DMTF for PLDM type 5 enhancements to make this feature available.

3.3 Firmware Image Format

Requirement: Firmware shall be made available in PLDM for firmware update package format, as described in DSP0267.

Here are key highlights of using this package format:

- The package format allows targeting one/several/all device types (for example, all GPUs, all Retimers, PCIe Switches, and so on). This provides the flexibility to update a specific device in a system, optimizing update time for complex GPU systems that need only one device updated. This also allows one package to update all devices in a system at one time.
- The package format accommodates a scenario where certain instances of a device require a firmware image, and the remaining instances of the device require a different firmware image.
- If a certain firmware image is applicable to N instances of a device type, the package will comprise only one copy of that image.
- The package format uses standard identifiers (such as UUID) to target device types.

3.4 Firmware Image Size

Firmware package shall not exceed 200MB in size.

3.5 Firmware Interchangeability

A vendor may fulfill orders of a particular product with physically different, yet interchangeable, units. These differences may require a different firmware to be installed on the nonvolatile storage for the firmware device storage. This specification extends the definition of interchangeability such that the same firmware image or firmware bundle provided for the product must support all interchangeable units.

NOTE: Product hardware may evolve over time for a variety of reasons, such as to improve reliability or react to supply chain deficiencies. The vendor is expected to notify the customer of future hardware changes as well as expected firmware changes. However, the validation and acceptance of physically different, yet interchangeable, units is outside the scope of this specification.

The requirement for a firmware image or firmware bundle to support all interchangeable units extends only to units with the same orderable part number. The goal of the requirement is to ensure that a particular instance can be managed without the disruption of new firmware update development when the underlying hardware changes. If the hardware changes are significant enough that the vendor no longer offers the same orderable part number, then a firmware image or firmware bundle that supports both the discontinued product and the replacement product is not required by this specification.

The firmware image or firmware bundle for interchangeable units must be backward compatible, but is not required to be forward compatible. A firmware image or firmware bundle is not required to support future interchangeable HW variations not known at the time of firmware release. However, the firmware image or firmware bundle is required to support all previous interchangeable HW variations known at the time of firmware release, including those that have been discontinued.

When different firmware is required for interchangeable units of the same product, the firmware update process for hyperscalers should remain unchanged. The firmware update process must include detection of the hardware and be able to select and install the corresponding firmware image. Firmware interchangeability may be implemented using either a proprietary design or a design based on a firmware update standard like DSP0267 (PLDM Type 5).

The PLDM Type 5 matching algorithm requires that all Device Descriptors in Firmware Device Record must match a descriptor returned by the QueryDeviceIdentifiers command. If a unique firmware image must be transferred to the firmware device for update, then the following are required:

- The set of descriptors returned by QueryDeviceIdentifiers must uniquely identify the variation between interchangeable units
- The firmware update package must have multiple Firmware Device Records, one for each of the uniquely identifiable interchangeable unit variations
- The Component Image information for the Firmware Device Record must describe and point to the corresponding firmware image for that interchangeable unit variation

Standard PLDM Type 5 device descriptors, such as PCI Device ID, may necessarily be the same between interchangeable unit variations. To describe the variations, one or more Vendor Defined descriptors may be used. Defining a specific set of descriptors is an implementation detail beyond the scope of this specification.

The ability for the update agent to detect which firmware image to send to the device is an optimization. The device itself should also check the target of the firmware image it receives and reject or fail the update if it is not correct for that unit variant. The information used by the device to perform the check must be trusted (signed) in order to satisfy the requirements for secure firmware update.

3.6 Single shot firmware update

The Firmware device Product shall support all firmware component images as part of the PLDM firmware update package as defined in the header structure in DSP0267. The BMC will process the firmware bundle and update the firmware in parallel using a single package.

The device may also provide the option to update one or more firmware in the firmware bundle as selected by the user. If implemented and if MultiPartHttpPushUri is not supported by the device, then **UpdateService must** support **HttpPushUriTargets.**

HttpPushUriTargets	Array	Read-write	An array of URIs that indicate where to apply the update image	Optional

Table 9: HttpPushUriTargets

3.7 Firmware Update Failure

If a Firmware Update operation of a FW bundle fails to copy/arm a subset of the devices in the package, the update operation will be considered a failure. This is because the consistency of the bundle would be broken and may not be guaranteed to work. An example of such a failure for a package targeting a UBB comprised of 2 GPUs and a FPGA is provided below

FW Device in package	FW Version in HW T=0 (current running Pkg N)	FW Version in HW T=1 (after Copy+Arm of Pkg N+1)	Scenario 1: FW Version in HW T=2 (retry Copy+Arm Pkg N+1)	Scenario 2: FW Version in HW T=2 (retry Copy+Arm Pkg N)	Scenario 3: FW Version in HW T=2 (return Copy+Arm Pkg N+1)
GPU0	X	X+1	X+1	X	X+1
GPU1	Y	Y	Y+1	Y	Y
FPGA	Z	Z+1	Z+1	Z	Z+1
FW Update Status	Running	Failed	Pass	Pass	Failed

Table 10: FW Update Failure scenarios

The example illustrates a case where the currently running version is package N consisting of version X for GPU0, version Y for GPU1 and version Z for the FPGA. At time T=1, the UBB is updated with package N+1. That results in GPU0 and FPGA to be successfully updated (Armed and ready to run those versions upon activation) but GPU1 fails to be updated and remains in version Y. This will result in the FW Update operation to fail at the interface API. The client can then retry the new package either successfully (Scenario 1) or with an error (Scenario 3). The client is also free to update with Package N (like a rollback to the last known good package) as shown in Scenario 2. If repeated attempts to update fail, then the client can choose to go down the recovery path as described in the next sections or treat the HW as needing repairs as per the CSP's policies.

A device reset/reboot/power cycle should not be required to retry a FW Update operation (Copy and/or Arm).

3.8 Firmware Image redundancy

Every firmware device shall support a minimum of 2 copies of firmware that the component can boot from. The Firmware Device can choose to implement any scheme of choice, e.g

Active/Backup, Active/Golden. Active/Known Good. Vendors may choose to implement redundancy using a second discrete EEPROM or a distinct partition on an EEPROM.

3.9 Firmware resiliency

3.9.1 Single Failure

Here are some reasons that firmware updates may fail:

- Transient failures that might occur due to bit flips that are the result of cosmic radiation, overheating issues, or errors that might occur during one boot cycle and go away during the next boot.
- Bit errors on packets that are sent over physical transport like I2C/SPI
- Hardware degradation.
- Any interruptions to the FW Update process such as Power loss, Machine Checks etc.

To address these failure modes, and minimize human intervention to recover from failed firmware updates, the vendor shall use the following principles and approaches:

1) Implement detection mechanism for FW update failure
2) Implement automatic switchover to the redundant copy of FW
3) Support FW Update telemetry in Section 3.10
4) Optionally, the GPU/Accelerator product may choose to automatically recover the failed image from the working image.

3.9.2 Double Failure

If, during the time interval between copying the image to the device and arming, the current firmware on the device crashes and fails to boot again (double fault), the BMC or AMC should be able to copy current or some working image to recover the device. This is a case of double fault where the active firmware image is rendered non-operational and the new FW image is still active as it has only been copied and not armed yet. This recovery mechanism is required since the device will not failover to the inactive image as the device is not armed yet.

3.10 Firmware telemetry

Firmware devices shall support querying of the following firmware telemetry:

1) FW version information of all images saved on each device
2) Information on which image device booted off of
3) Firmware Device Boot Result (succ/auth. _fail/boot_fail)
4) Firmware Device Update Result (succ/auth. _fail/boot_fail)
5) Firmware Device Fallback Reason (auth._fail/boot_fail)
6) Activation options
7) Activation status
8) Flash write-protect state

3.11 Debug messages for runtime DC Ops

Vendor shall provide clear actionable debug messages for status and error conditions:

- All logs must be in plain text, not encrypted.
- Clear Actionable Status Codes for FW Update operations (e.g. state of progress, recoverable errors, failures)

4. Redfish Update Service

4.1 Common properties

The following properties are defined for inclusion in every Redfish schema, and therefore may be encountered in any response payload. They are documented here to avoid repetition in the response property tables and must be included in every Redfish API response.

Property	Type	Attribute	Notes
@odata.id	string	Read-only	The unique ID for the resource and it shall be of the form defined in the Redfish Specification (DSP0266). For UpdateService, it shall be "/redfish/v1/UpdateService"
@odata.type	string	Read-only	The type of a resource and it shall be of the form defined in the Redfish specification (DSP0266). For UpdateService, , it shall be "#UpdateService.<Version>.UpdateService"

Description	string	Read-only	Human readable description for the resource.
Id	string	Read-only	The ID that uniquely identifies the Resource within the collection that contains it. For UpdateService, it shall be “UpdateService”
Name	string	Read-only	The human-readable moniker for a Resource. The type is string. The value is NOT necessarily unique across Resource instances within a collection.

Table 11: GPU FW update Redfish API properties

4.2 Update Service and Software Inventory

4.2.1 UpdateService

An UpdateService resource should provide a response that describes the update service and the properties for the service itself with links to collections of firmware and software inventory. The update service also provides methods for updating software and firmware of the resources in a Redfish service.

The following tables contain the requirements of the UpdateService resource.

Property	**Type**	**Attribute**	**Notes**	**Required/** **Optional**

FirmwareInventory	Object		An inventory of firmware	Required
HttpPushUri	String	Read-only	The URI used to perform an HTTP or HTTPS push update to the update service.	Required for legacy one-shot support. Optional if MultipartHttpPushUri is supported
HttpPushUriOptions	Object		The options for HttpPushUri provided software updates	Required for legacy one-shot support
HttpPushUriOptions/HttpPushUriApplyTime	Object		The settings for when to apply `HttpPushUri`-provided software.	Required for legacy one-shot support
HttpPushUriOptions/HttpPushUriApplyTime/ApplyTime	String		The time when to apply the `HttpPushUri`-provided software update	Required for legacy one-shot support
HttpPushUriOptionsBusy	Boolean	Read-write	An indication of whether a client has reserved the HttpPushUriOptions	Optional for legacy

			properties for software updates	one-shot support
HttpPushUriTargets	Array	Read-write	An array of URIs that indicate where to apply the update image	Optional for legacy one-shot support
HttpPushUriTargetsBusy	Boolean	Read-write	An indication of whether any client has reserved the HttpPushUriTargets property	Optional for legacy one-shot support
MaxImageSizeBytes	Integer	Read-only	The maximum size in bytes of the software update image that the service supports	Required
MultipartHttpPushUri	String	Read-only	The URI used to perform a Redfish specification defined Multipart HTTP or HTTPS push update to the update service	Optional for One-Shot but Required for Split Copy/ARM support
ServiceEnabled	Boolean	Read-write	An indication of whether this service is enabled	Required
SoftwareInventory	Object		An inventory of software	Required

Table 12: Redfish UpdateService property

The Redfish service can optionally support StartUpdate actions. The requirements of action parameters are as follows.

Actions / Description	**Parameter**	**Required/ Optional**
StartUpdate – This action starts updating all images that have been previously invoked using an OperationApplyTime value of OnStartUpdateRequest.	None	Optional

Table 13: Redfish UpdateService actions

Example:

An example is given below. A good reference can also be found in DMTF's Redfish FW Update White Paper (DSP2062) section 3.1
https://www.dmtf.org/sites/default/files/standards/documents/DSP2062_1.0.2.pdf

```
{
      "@odata.type": "#UpdateService.v1_11_0.UpdateService",
      "@odata.id": "/redfish/v1/UpdateService",
      "Id": "UpdateService",
      "Name": "Update Service",
      "HttpPushUri": "/redfish/v1/UpdateService/update",
      "HttpPushUriTargets": [],
      "HttpPushUriOptions": {
          "ForceUpdate": false,
          "HttpPushUriApplyTime": {
              "ApplyTime": "OnReset"
          }
```

```
      },
      "MaxImageSizeBytes": 1024000,
      "ServiceEnabled": true,
      "FirmwareInventory": {
          "@odata.id":
"/redfish/v1/UpdateService/FirmwareInventory"
      },
      "SoftwareInventory": {
          "@odata.id":
"/redfish/v1/UpdateService/SoftwareInventory"
      }
}
```

4.2.2 Software Inventory

The FirmwareInventory and SoftwareInventory properties contain a link to the collection of SoftwareInventory resources.

The SoftwareInventory resource collection contains the set of software components, generally like drivers execute within a host operating system. Software in this collection is generally updated using operating system-specific centric methods.

The FirmwareInventory resource collection contains the set of firmware components generally referred to as platform firmware or that does not execute within a host operating system. Software in this collection is generally updated using platform-specific methods or utilities.

Example SoftwareInventory resource:

```
{
  "@odata.type": "#SoftwareInventory.v1_4_0.SoftwareInventory",
  "@odata.id":
"/redfish/v1/UpdateService/FirmwareInventory/ManagementController"
,
  "Id": "ManagementController",
  "Name": "ManagementController Firmware",
  "Status": {
    "State": "Enabled",
```

```
    "Health": "OK"
  },
  "SoftwareId": "0x1000",
  "Manufacturer": "Vendor",
  "Updateable": true,
  "WriteProtected": false,
  "Version": "1.0.0",
  "RelatedItem": [
    {
      "@odata.id": "/redfish/v1/Chassis/ManagementController"
    }
  ]
}
```

The following table contains the requirements of the SoftwareInventory resources.

Property	**Type**	**Attribute**	**Notes**	**Required/ Optional**
Id	String	Read Only	Name of the firmware component as used in the URI/@odata.id	Required
Name	String	Read Only	A name that gives better clarity on the firmware component	Required
Status.State	String	Read Only	The state of the firmware component. Allowable Values : “Enabled” ”Disabled”	Required

Status.Health	String	Read Only	The health of the firmware component Allowable Values : “OK” ”Warning” “Critical”	Required
SoftwareId	String	Read Only	Reflects the ComponentIdentifier of the PLDM header. The String should reflect the UINT16 value like this: “0x1000”	Required
Manufacturer	String	Read Only	Name of the manufacturer	Required
Updateable	Boolean	Read Only	Reflects update capability of the component	Required
WriteProtected	Boolean	Read Only	True only if the component is write protected	Required
Version	String	Read Only	Matches the ComponentVersionString of the PLDM header	Required

Table 13: Redfish Software Inventory

4.3 TaskService

4.3.1 Task

Task provides asynchronous capability to track the progress of the Firmware update progress including status, progress.

Property	**Type**	**Attribute**	**Notes**	**Required/ Optional**
TaskState	String	Read-only	This property shall indicate the state of the task Allowed values are · "New" · "Starting" · "Running" · "Suspended" · "Interrupted" · "Pending" · "Stopping" · "Completed" · "Killed" · "Exception" · "Cancelled"	Required

TaskStatus	String	Read-Only	This shall be set only up on task completion “OK” - If all updates successfully completed “Warning” - If one or more of the updates completed with warnings “Critical” if one or more of the updates had critical severity failures	Required
StartTime	String	Read-only	The date and time when the task was started.	Required
EndTime	String	Read-only	The date and time when the task was completed. This property will only appear when the task is complete.	Required

TaskStatus	Boolean	Read-only	The completion status of the task. Allowed values are · "Critical" · "Warning" · "OK"	Required
PercentComplete	Number	Read-only	The completion percentage of this task.	Required
Messages	Object	Read-only	This property shall contain an array of messages associated with the task.	Required
TaskMonitor	String	Read-only	The URI of the Task Monitor for this task.	Required

Table 15: Redfish Task Service property

4.4 Redfish Error Reporting for Firmware Update

Firmware update processes can run into several types of failures. This section contains various error scenarios and fault descriptions that the device should support.
Note: The versions in the tables in this section refer to minimum versions that must be supported. A device may suppose higher versions based on the registry definitions here https://redfish.dmtf.org/registries. The actual Message strings (e.g. "UpdateInProgress") must be supported as described in the tables below.

4.4.1 Message Registry

Devices shall support DMTF Redfish standards defined Message registry (as defined in DSP2065).

Property	**Type**	**Attribute**	**Notes**	**Required/ Optional**
@odata.type	String	Read-only	This property shall indicate the value of the type property for the Message object. "#Message.<version>.Message"	Required
MessageId	String	Read-Only	Property as defined in the Redfish specification.	Required
Message	String	Read-Only	Property contains human readable message string	Required
MessageArgs	Collection	Read-Only	Property contains an array of arguments to	Optional

			support qualify the message	
Severity	String	Read-Only	Property indicates the severity of the failure. Supported “OK”, “Warning”, “Critical”	Required
Resolution	String	Read-Only	Property contains the device indicated recommendation on recovering from the failure or next steps	Required

Table 16: Redfish Message Registry

4.4.2 Immediate Failures

Immediate failures are types of failures, where the device responds back immediately to the Orchestrator as response to invoking with the UpdateService URI. No background task monitoring is started due to the nature of error.
Device shall support the following immediate failure modes if they occur before the Redfish Task is generated

Failure mode	**HTTP error code**	**Min MessageID Version**	**Other details (Only example Resolution shown below)**	**Supplier collateral needed for compliance check ?**

Another Firmware Update already in progress	400	**Update.1.0.0.UpdateInProgress**	Message:" An update is in progress." MessageArgs: None Resolution: "Wait for the previous operation to be completed."	N
Incoming file is not PLDM file format	500	**Base1.8.1.ResourceAtUriInUnknownFormat**	Message: "The resource at '<Arg1>' is in a format not recognized by the service." MessageArgs: "filename of the uploaded file" Resolution: "Upload a file supported by this device"	N
Insufficient Storage Space (*)	500	**Base.1.13.0.InsufficientStorage** **NOTE:** Need a scenario (use-case) to generate this failure. Are we referring to RAM/NV memory?	Message: "Insufficient storage or memory available to complete the request." MessageArgs: None Resolution: "Reset the baseboard and retry the operation."	TBD
Incoming file exceeds max size supported	500	**Base.1.13.0.PayloadTooLarge**	Message: "The supplied payload exceeds the maximum size supported by the service." MessageArgs: None Resolution: "Firmware package is greater than allowed size. Check the file before continuing"	Yes [Supplier to provide a bundle which is signed but too large]
NVRAM ownership failed (SPI, ..)	200	**ResourceEvent.1.0.0.ResourceErrorsDetected**	Message: "The resource property 'Component Name' has detected errors of type 'SPI Access Error'."." MessageArgs: ["Component Name", "SPI Access Error"] Resolution: "Retry few times, reset the device and retry"	NA
Internal Error (all other exceptions)	500	**Base.1.0.0.InternalError**	Message: "The request failed due to an internal service error. The service is still operational" MessageArgs: None Resolution: "Resubmit the request.	N/A

			If the problem persists, consider resetting the service”	

Table 17: Redfish Immediate Failure codes

4.4.3 Failure during Asynchronous Operation

Once the device accepts the Update Service Firmware update command successfully, it returns a Task ID. The orchestrator keeps monitoring the task for completion. Completion can result in failure scenarios. Devices that support multiple firmware components update in a single bundle, each firmware component can return an individual completion (success or failure). In all these error cases, the task will return TaskState=Exception and TaskStatus=Critical.

Device shall support the following async failure modes if they occur after the Redfish Task is generated.

Failure mode	**HTTP code**	**MessageID**	**Other details**	**Supplier collateral needed for compliance check ?**
NVRAM ownership failed (SPI, ..)	200	**ResourceEvent.1.0.0.ResourceErrorsDetected**	Message: ”The resource property 'Component Name' has detected errors of type 'SPI Access Error'.”,” MessageArgs: [“Component Name”, “SPI Access Error”] Resolution: “Retry few times, reset the device and retry”	NA
Incoming file is not PLDM file format	500	**Base1.8.1.ResourceAtUriInUnknownFormat**	Message: ”The resource at ‘<Arg1>’ is in a format not recognized by the service.” MessageArgs: “filename of the uploaded file” Resolution: ”Upload a file supported by this device”	N

PLDM bundle meta-data corruption*		**Update.1.0.0.VerificationFailed** **NOTE:** What is deemed as meta data corruption	Message:"Verification of image '<Arg1>' at '<Arg2> ' failed." MessageArgs: [<Arg1> - Identifier for the image (Ex: CRC/hash, SKU/FRU) <Arg2> - Name of the target verifying the image (Ex: eRoT)] Resolution: None	
PLDM Bundle signature check failure		**Update.1.0.0.VerificationFailed**	Message:"Verification of image '<Arg1>' at '<Arg2> ' failed." MessageArgs: [<Arg1> - "PLDM Bundle" <Arg2> - Name of the target verifying the image (Ex: eRoT)] Resolution: None	N
Component verification failure (before APPLY)		**Update.1.0.0.VerificationFailed**	Message:"Verification of image '<Arg1>' at '<Arg2> ' failed." MessageArgs: [<Arg1> - FW component Name; <Arg2> - Name of the target verifying the image (Ex: eRoT)] Resolution: None	

* Details for these failure modes will be defined in the next version of the specification

Table 18: Background Failure codes

4.4.4 Failure during activation/background-copy operation

This section assumes that staging of the PLDM bundle is complete and only the failures during FW activation or background copy operation are captured here. In all these failure cases, the device shall send a Redfish event to the orchestrator with Severity as 'Critical'.

Device shall support the following async failure modes if they occur after the Redfish task is generated.

Failure mode	HTTP code	MessageID	Other details	Supplier collateral needed for compliance check ?
NVRAM ownership failed (SPI, ..)	200	**ResourceEvent.1.0.0.ResourceErrorsDetected**	Message: ”The resource property 'Component Name' has detected errors of type 'SPI Access Error'.",” MessageArgs: [“Component Name”, “SPI Access Error”] Resolution: “Retry few times, reset the device and retry”	NA for 1.1
Activation failure *		**Update.1.0.0.ActivateFailed** **Scenarios:** Failure to TX the N+1 IFWI to IFWI flash N+1 verification failure at device level Update.1.0.1.ActivateFailed	Message: ”Activation of image '<Arg1>' on '<Arg2>' failed” MessageArgs: [Arg1: FW image name Arg2: FW device] Resolution: “Retry a few times. If persists, Do not update to new FW”	NA for 1.1
Background copy of PLDM bundle failure*		Can we use **Update.1.0.0.TransferFailed** with relevant Arg1 and Arg2 ? NOTE: Do we need a specific message ID in the future Update registry ?	Message: ”Transfer of image '<Arg1>' to '<Arg2>' failed” MessageArgs: [Arg1: Active PLDM bundle version Arg2: Redundant partition] Resolution: “Retry a few times. If persists, Do not update to new FW”	NA for 1.1

4.5 Redfish API Examples for Multipart Push Update and Split Copy/Arm support

To support Redfish APIs for the manual firmware Arming option, we are proposing Oem extensions to the MultipartHttpPushUri-provided firmware updates.

The following workflow shows the Redfish API examples for performing multipart push update and allows hyperscalers to opt for manual firmware Arming option, delaying Arming until a specified action is initiated by the client.

1.Query `MultipartHttpPushUri` property within the UpdateService and `DelayedFirmwareArmingSupport` property for the delayed Arming options provided by the service.

Example UpdateService resource:

```
{
    "@odata.type": "#UpdateService.v1_11_0.UpdateService",
    "@odata.id": "/redfish/v1/UpdateService",
    "Id": "UpdateService",
    "Name": "Update Service",
    "MultipartHttpPushUri": "<URI used to perform a multipart push
update>",
    "MaxImageSizeBytes": 1024000,
    "ServiceEnabled": true,
    "FirmwareInventory": {
        "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory"
    },
    "Oem": {
        "OCPGPUMgmtWG":{
            "@odata.type":
"#OpenCompute_GPUMgmtIntfUpdateService.v0_7_0.OpenCompute_GPUMgmtI
ntfUpdateService",
            "MultipartHttpPushUriOptions": {
                "DelayedFirmwareArmingSupport": ["None",
"InitiateOnArmFirmwareRequest"]
            }
        }
    },
    "Actions": {
        "Oem":{
```

```
            "#OpenCompute_GPUMgmtIntfUpdateService.ArmFirmware": {
                "target":
"/redfish/v1/UpdateService/Actions/Oem/OpenCompute_GPUMgmtIntfUpda
teService.ArmFirmware"
            }
        }
    }
}
```

2.Transfer/Copy the image to the target devices using `MultipartHttpPushUri` and select a desired `DelayedFirmwareArming` update parameter.

- For the automatic firmware Arming option, select `None` value for the `DelayedFirmwareArming` parameter or skip this parameter to push the image to the target devices by using the URI provided by MultipartPushUri.

  Example:

  ```
  POST /redfish/v1/UpdateService/update-multipart HTTP/1.1
  Content-Type: multipart/form-data;
  boundary=--------------------------d74496d66958873e
  Content-Length: <computed-length>
  ----------------------------d74496d66958873e
  Content-Disposition: form-data; name="UpdateParameters"
  Content-Type: application/json
  {}
  ----------------------------d74496d66958873e
  Content-Disposition: form-data; name="UpdateFile";
  filename="image.bin"
  Content-Type: application/octet-stream
  <software image binary>
  ```

- For manual firmware Arming option, select `InitiateOnArmFirmwareRequest` value for the `DelayedFirmwareArming` parameter to push the image to the target devices by using the URI provided by the MultipartPushUri.

  Example:

  ```
  POST /redfish/v1/UpdateService/update-multipart HTTP/1.1
  Content-Type: multipart/form-data;
  boundary=--------------------------d74496d66958873e
  ```

```
Content-Length: <computed-length>
----------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateParameters"
Content-Type: application/json
{
    "Oem": {
        "OpenCompute_GPUMgmtIntf":{
            "@odata.type":
"#OpenCompute_GPUMgmtIntfUpdateService.v0_7_0.OpenCompute_GPU
MgmtIntfUpdateService",
            "DelayedFirmwareArming":
"InitiateOnArmFirmwareRequest"
        }
    }
}
----------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateFile";
filename="image.bin"
Content-Type: application/octet-stream
<software image binary>
```

3. The client transitions to monitoring the task for firmware update progress. Refer to examples in the Redfish Firmware Update White Paper (DSP2062).

4. The client verifies `PendingArming` value of the `FirmwareState` property in the `InactiveFirmwareSlot` object to confirm the image had been successfully copied to the target device, and awaiting for a client action to Arm.

Example FirmwareInventory resource:

```
{
  "@odata.type": "#SoftwareInventory.v1_4_0.SoftwareInventory",
  "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/GPU0",
  "Id": "GPU0",
  "Name": "GPU0 Firmware",
  "Status": {
    "State": "Enabled",
    "Health": "OK"
  },
  "SoftwareId": "0x1000",
```

```
  "Manufacturer": "<vendor>",
  "Updateable": true,
  "WriteProtected": false,
  "Version": "<firmware version>",
  "Oem": {
    "OpenCompute_GPUMgmtIntf":{
      "@odata.type":
"#OpenCompute_GPUMgmtIntfSoftwareInventory.v0_7_0.OpenCompute_GPUM
gmtIntfSoftwareInventory",
      "DelayedFirmwareArmingSupported": true,
      "ActiveFirmwareSlot": {
        "Version" : "1.0.0",
        "FirmwareState": "Activated"
      },
      "InactiveFirmwareSlot": {
        "Version" : "2.0.0",
        "FirmwareState": "PendingArming"
      }
    }
  }
}
```

5. Arm the copied firmware for next activation by performing an HTTP POST to the target URI of the `ArmFirmware` action.

ArmFirmware action

- This action is used to Arm all images that have been previously staged using a DelayedFirmwareArming value of InitiateOnArmFirmwareRequest.

Action URI

- {Base URI of target resource}/Actions/OpenCompute_GPUMgmtIntfUpdateService.ArmFirmware

Example:

```
POST
/redfish/v1/UpdateService/OpenCompute_GPUMgmtIntfUpdateService.Arm
Firmware
{
      "Targets": ["/redfish/v1/Managers/1"]
}
```

Note: Targets array is an optional parameter. If this parameter is not present or contains no targets, the service shall apply to all applicable targets.

5. PLDM Update

5.1. PLDM Update Commands

The table below indicates the set of PLDM commands that should be supported by all GPUs as part of the Firmware Update & Device Inventory discovery commands:

Device Inventory Discovery Commands	
Command Code	Command Name
0x01	QueryDeviceIdentifiers
0x02	GetFirmwareParameters
Firmware Update Commands	
Command Code	Command Name
0x10	RequestUpdate

0x11	GetPackageData (Optional*)
0x12	GetDeviceMetaData (Optional*)
0x13	PassComponentTable
0x14	UpdateComponent
0x15	RequestFirmwareData
0x16	TransferComplete
0x17	VerifyComplete
0x18	ApplyComplete
0x19	GetMetaData (Optional*)
0x1A	ActivateFirmware
0x1B	GetStatus

0x1C	CancelUpdateComponent
0x1D	CancelUpdate
0x21	GetComponentOpaqueData (Optional*)

Table 19: PLDM Update commands

Optional* :- The commands which are marked as optional since it depends on the Firmware Update Package format and the capabilities from the device that is supported.

Requests and response format for the above commands should be of the same format as DSP0267 v1.0.0 specification. The request and response formats for the commands are present in Appendix below.

The “CompletionCode” returned in the response should be of the format of PLDM_BASE_CODES defined according to DSP0240 v1.1.0 specification. In addition, PLDM completion codes specific for firmware update that are beyond the scope of PLDM_BASE_CODES should be supported according to the format as defined in DSP0267 v1.0.0 specification.

5.2 PLDM Update Timing Requirements

The timings as specified in DSP0246 v1.2.0 specification needs to be supported from the endpoints.

Timing	**Min**	**Max**	**Description**

Number of request retries when a response is received that requires a retry	2s		Total of three tries, minimum: the original try plus two retries.
Update mode idle timeout	60s	120s	Amount of time before the FD/FDP shall exit from update mode if no command is received from the Update Agent when it's expected, during the firmware update process
Retry request for firmware data	1s	5s	Amount of time for the FD/FDP 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	500ms	5s	Amount of time to wait before the UA sends an additional CancelUpdate or CancelUpdateComponent command.
Request firmware data idle timeout	60s	90s	Amount of time for the Update Agent to cancel the component update if no command is received from the FD/FDP when it's expected, during the component image transfer stage.
State change timeout	180s		Amount of time for the Update Agent to wait before canceling the component update if the ProgressPercent value in the GetStatus command remains unchanged

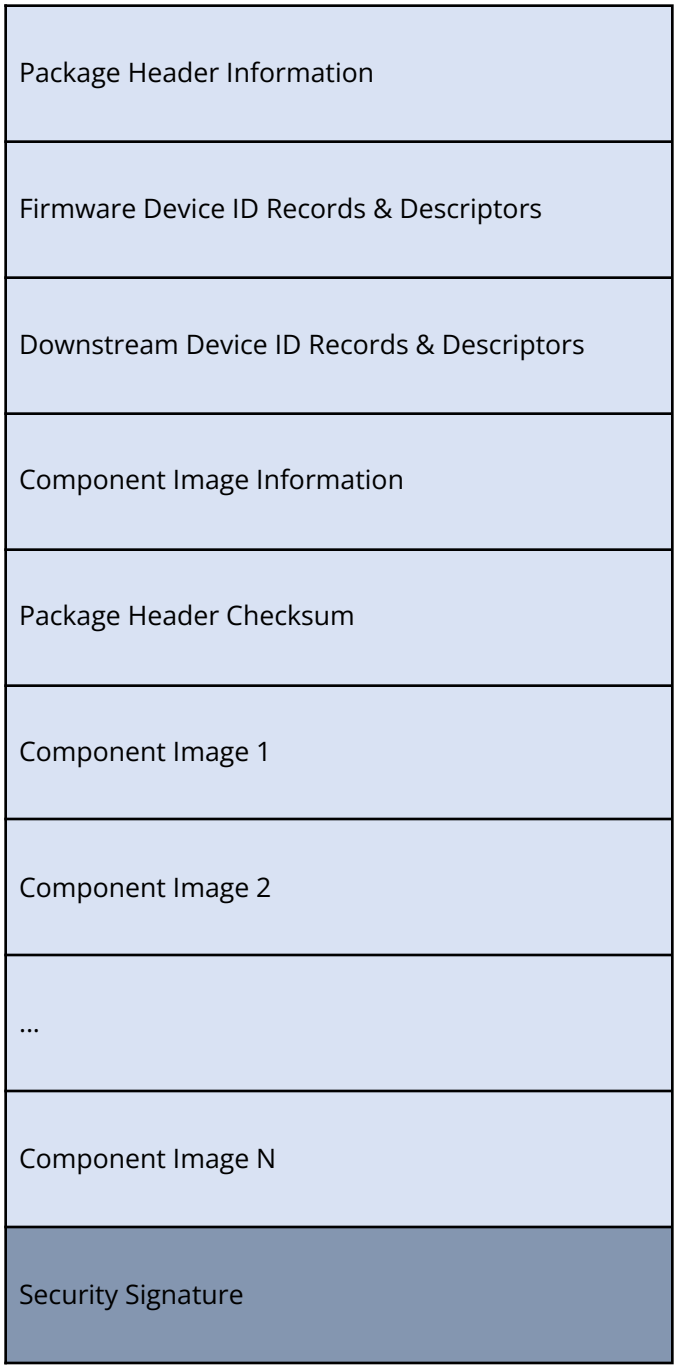
Retry request for update	1s	5s	Amount of time for the UA to wait before resending a RequestUpdate or RequestDownstreamDevice Update command after receiving a RETRY_REQUEST_UPDATE code from the FD/FDP
Get Package Data timeout	1s	5s	Amount of time for the UA to wait to receive the GetPackageData command if the FD/FDP indicated that it would send that command in the response to RequestUpdate or RequestDownstreamDeviceUpdate. The UA shall send CancelUpdate if this timer expires
Complete Commands Timeout	600s		Amount of time for the UA to wait for a TransferComplete, VerifyComplete, or ApplyComplete command if the ProgressPercent value in the GetStatus command is set to 0x65 (not supported by FD/FDP).

Table 20: PLDM Update timing requirements

6. File Format

Supported file format for Redfish Update Service is DMTF PLDM File format as described in DMTF DSP0267 v1.0.

6.1 File Format


Package Header Information
Firmware Device ID Records & Descriptors
Downstream Device ID Records & Descriptors
Component Image Information
Package Header Checksum
Component Image 1
Component Image 2
...
Component Image N
Security Signature

6.2 Package Security

This specification enhances the requirement of DSP0267 from making signing of the file from optional to mandatory. The entire file needs to be signed with ECDSA-P384+SHA-384 algorithm.

The entire bundle needs to be enveloped with a Security signature appended at the end of the file. Security signature shall contain

- Security versions
- Device binding
- Keys/Hashes of key
- Signature

Each Firmware component in the file bundle is individually signed with ECDSA-P384+SHA-384 algorithm. Each component verifies the signature during the firmware update process.

7. References

1. “Redfish Firmware Update White Paper”, https://www.dmtf.org/dsp/DSP2062

8. License

There are 2 types of license under which a document can be submitted. Please provide one only.

- Creative Commons - This is the suggested language.

  OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

  It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, “Materials”) are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: https://creativecommons.org/licenses/by-sa/4.0/, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

  As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP’s request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at http://www.opencompute.org/participate/legal-documents/. If you have any questions please contact OCP.

  Footer:

  This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- Copyright License - This is the suggested language.

© Copyright [Company Name][Year] All rights reserved.

If your company has different copyright language, please insert your copyright information here.

OCP will need a signed Copyright License Agreement so we can publish this on our website. Please insure you have signed this agreement.

About Open Compute Foundation

At the core of the Open Compute Project (OCP) is its Community of hyperscale data center operators, joined by telecom and colocation providers and enterprise IT users, working with vendors to develop open innovations that, when embedded in product are deployed from the cloud to the edge. The OCP Foundation is responsible for fostering and serving the OCP Community to meet the market and shape the future, taking hyperscale led innovations to everyone. Meeting the market is accomplished through open designs and best practices, and with data center facility and IT equipment embedding OCP Community-developed innovations for efficiency, at-scale operations and sustainability. Shaping the future includes investing in strategic initiatives that prepare the IT ecosystem for major changes, such as AI & ML, optics, advanced cooling techniques, and composable silicon. Learn more at www.opencompute.org.

Appendix A. [Redfish API Example]

The examples here are only for illustrative purposes and not meant to be a code reference for implementation. For the source of truth for implementation please refer to the latest DMTF Redfish Firmware Update White Paper (DSP2062) and Redfish Specification (DSP0266)

1. Initiate a firmware update for all devices on the baseboard.

You must apply the firmware update package released by Vendor as is.

a. Determine where to perform an `HTTP POST` of the firmware update package. The `URI` to which the package is to be `POST`ed is in the `HttpPushUri` property.

```
curl http://${amc}/redfish/v1/UpdateService
{
   "@odata.id": "/redfish/v1/UpdateService",
   "@odata.type": "#UpdateService.v1_4_0.UpdateService",
   "Description": "Service for Software Update",
   "FirmwareInventory": {
         "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory"
   },
   "HttpPushUri": "/redfish/v1/UpdateService/update",
   "HttpPushUriOptions": {
         "HttpPushUriApplyTime": {
               "ApplyTime": "OnReset"
         }
   }
   "Id": "UpdateService",
   "Name": "Update Service",
   "ServiceEnabled": true
}
```

b. Push the firmware update package to the AMC using `HTTP POST`:

```
curl -X POST -T <firmware update package>
http://${amc}/redfish/v1/UpdateService/update
```

See step 3 for information about the response to this request.

2. Initiate the firmware update for specific devices on the baseboard.

a. Specify the set of URIs to which the firmware update package should be targeted. These `URI`s will be passed to the `HttpPushUriTargets` property. The URIs should be of FirmwareInventory resources. Here is an example where only GPU firmware will be updated:

```
curl -X PATCH -d '{ "HttpPushUriTargets":[
"/redfish/v1/UpdateService/FirmwareInventory/DEVICE_1",
"/redfish/v1/UpdateService/FirmwareInventory/DEVICE_2",
"/redfish/v1/UpdateService/FirmwareInventory/DEVICE_3",
"/redfish/v1/UpdateService/FirmwareInventory/DEVICE_4",
]}' http://${amc}/redfish/v1/UpdateService
```

b. Repeat step 1b.

Note: The firmware update package needs to be pushed (`HTTP POST`ed) only once, regardless of the number of URIs in `HttpPushUriTargets`. The `HttpPushUriTargets` allows users to select a subset of firmware components for an update after the DSP0267 package matching. Clearing HttpPushUriTargets allows users to do firmware updates according to DSP0267 package matching.

3. Monitor the firmware update progress.

Redfish-based firmware update is an asynchronous operation (see DSP0266). The AMC's Redfish service will implement the `TaskService` schema and return a task monitor `URI` from the Location header and a `Task` representation as the response to the `HTTP` post and a `202` Accepted `HTTP` response code:

```
HTTP/1.1 202 Accepted
Location: /redfish/v1/TaskService/Tasks/1/Monitor
Retry-After: 30
Content-Type: application/json
{
    "@odata.id": "/redfish/v1/TaskService/Tasks/1",
    "@odata.type": "#Task.v1_4_3.Task",
    "Id": "1",
```

```
        "TaskState": "Running",
        "TaskStatus": "OK"
    }
```

The `Task URI` in odata.id will implement the Task schema and use the task URI to monitor the progress of the firmware update. Here are the key properties of the task resource:

- `TaskState`

Indicates current state of the task, which can be Running, Stopping, Completed, Exception, or Canceled.

- `TaskStatus`

This property will indicate completion status of the task. After the task completes (success/fail), this property will show the completion status of the task, which can be OK, Warning, or Critical.

- `Messages` array

The Messages array in the Task resource will contain messages that a Redfish client can use to track progress and to complete actions, such as a Reset operation, where user intervention is required. Step 4 lists the key Message Registry entries that the AMC will add to the Messages array for the firmware update operation. A Redfish client can poll the Task to retrieve the task progress information, such as TaskState, PercentComplete, and Messages.

- Task Resource Response examples

Task Resource Response: When the task is running

```
{
  "@odata.id": "/redfish/v1/TaskService/Tasks/1",
  "@odata.type": "#Task.v1_4_3.Task",
  "Id": "1",
  "Messages": [
    {
      "@odata.type": "#Message.v1_0_0.Message",
      "Message": "The task with id 1 has started.",
      "MessageArgs": [
        "1"
      ],
      "MessageId": "TaskEvent.1.0.1.TaskStarted",
```

```
      "Resolution": "None.",
      "Severity": "OK"
    },
    {
      "@odata.type": "#Message.v1_4_1.Message",
      "Message": "The target device 'FW_AMC_0' will be updated with image
'<The identifier for the image.>",
      "MessageArgs": [
        "FW_AMC_0",
        "<The identifier for the image.>"
      ],
      "MessageId": "Update.1.0.TargetDetermined",
      "Resolution": "None.",
      "Severity": "OK"
    },
    {
      "@odata.type": "#Message.v1_4_1.Message",
      "Message": "Image '<The identifier for the image.>' is being
transferred to 'FW_AMC_0'.",
      "MessageArgs": [
        "<The identifier for the image.>",
        "FW_AMC_0"
      ],
      "MessageId": "Update.1.0.TransferringToComponent",
      "Resolution": "None.",
      "Severity": "OK"
    }
  ],
  "Name": "Task 1",
  "PercentComplete": 0,
  "StartTime": "2023-03-16T02:18:38+00:00",
  "TaskMonitor": "/redfish/v1/TaskService/Tasks/1/Monitor",
  "TaskState": "Running",
  "TaskStatus": "OK"
}
```

Task Resource Response: When the task is completed.


```
{
  "@odata.id": "/redfish/v1/TaskService/Tasks/1",
  "@odata.type": "#Task.v1_4_3.Task",
  "EndTime": "2023-03-16T02:27:31+00:00",
  "Id": "1",
  "Messages": [
    {
      "@odata.type": "#Message.v1_0_0.Message",
      "Message": "The task with id 1 has started.",
      "MessageArgs": [
        "1"
      ],
      "MessageId": "TaskEvent.1.0.1.TaskStarted",
      "Resolution": "None.",
      "Severity": "OK"
    },
    {
      "@odata.type": "#Message.v1_4_1.Message",
      "Message": "The target device 'FW_AMC_0' will be updated with image
'<The identifier for the image.>'.",
      "MessageArgs": [
        "FW_AMC_0",
        "<The identifier for the image.>"
      ],
      "MessageId": "Update.1.0.TargetDetermined",
      "Resolution": "None.",
      "Severity": "OK"
    },
    {
      "@odata.type": "#Messagev1_4_1.Message",
      "Message": "Image '<The identifier for the image.>' is being
transferred to 'FW_AMC_0'.",
      "MessageArgs": [
```

```
        "<The identifier for the image.>",
        "FW_AMC_0"
      ],
      "MessageId": "Update.1.0.TransferringToComponent",
      "Resolution": "None.",
      "Severity": "OK"
    },
    {
      "@odata.type": "#Message.v1_4_1.Message",
      "Message": "Device 'FW_AMC_0' successfully updated with image '<The
identifier for the image.>'.",
      "MessageArgs": [
        "FW_AMC_0",
        "<The identifier for the image.>"
      ],
      "MessageId": "Update.1.0.UpdateSuccessful",
      "Resolution": "None.",
      "Severity": "OK"
    },
    {
      "@odata.type": "#Message.v1_4_1.Message",
      "Message": "Awaiting for an action to proceed with activating image
'<The identifier for the image.>' on 'FW_AMC_0'.",
      "MessageArgs": [
        "<The identifier for the image.>",
        "FW_AMC_0"
      ],
      "MessageId": "Update.1.0.AwaitToActivate",
      "Resolution": "System reboot or AC power cycle",
      "Severity": "OK"
    },
    {
      "@odata.type": "#Message.v1_0_0.Message",
      "MessageId": "TaskEvent.1.0.1.TaskCompletedOK",
      "Message": "The task with id 1 has Completed.",
```

```
      "MessageArgs": [
        "1"
      ]
      "Resolution": "None",
      "Severity": "OK"
    }
  ],
  "Name": "Task 1",
  "PercentComplete": 100,
  "StartTime": "2023-03-16T02:18:38+00:00",
  "TaskState": "Completed",
  "TaskStatus": "OK"
}
```


4. Firmware update-related Redfish messages.

Table 33 contains key Redfish messages that the Redfish Task, which corresponds to a firmware update, can contain. These messages are based on standard Redfish Message Registries. The Resolution property contains information about actions that a Redfish user might need to perform.

Table 33: Key Redfish Messages

Message Registry	MessageId	Description	Resolution	Expected Client Reaction
Update	Target Determined	Indicates that a target resource or device for a image has been determined for update	NA	Continue to monitor the update progress.
Update	TransferringTo Component	Indicates that the service is transferring an image to a component.	NA	Continue to monitor the update progress.

Update	Update Successful	Indicates that a resource or device was updated.	NA	No further actions needed.
Update	AwaitTo Activate	Indicates that the resource or device is awaiting for an action to proceed with activating an image.	Device specific action - for example, perform DC cycle of the baseboard.	Perform the requested action to advance the update operation
Update	Transfer Failed	Indicates that the service failed to transfer an image to a component.	Look at other messages to determine corrective actions	Look at other messages to determine corrective actions
Update	Verification Failed	Indicates that the component failed to verify an image.	Look at other messages to determine corrective actions	Look at other messages to determine corrective actions
Update	ApplyFailed	Indicates that the component failed to apply an image	Look at other messages to determine corrective actions	Look at other messages to determine corrective actions
Update	Activate Failed	Indicates that the component failed to activate the image.	Look at other messages to determine corrective actions	Look at other messages to determine corrective actions

5. Check Inventory after update is complete

After a firmware update and subsequent firmware activation, the Redfish client can look at the SoftwareInventory resources and validate the updated version and other information about the software inventory. Example of querying the firmware version of `DEVICE_1`.

```
curl
http://${amc}/redfish/v1/UpdateService/FirmwareInventory/DEVICE_1
{
    "@odata.id":
"/redfish/v1/UpdateService/FirmwareInventory/DEVICE_1",
    "@odata.type": "#SoftwareInventory.v1_1_0.SoftwareInventory",
    "Id": "DEVICE_1",
    "Name": "Software Inventory",
    "RelatedItem": [
        {
            "@odata.id": "/redfish/v1/Chassis/DEVICE_1"
        }
    ],
    "RelatedItem@odata.count": 1,
    "Status": {
        "HealthRollup": "OK",
        "State": "Enabled"
    },
    "Updateable": true,
    "Version": "XX.YY.ZZ"
}
```

6. To get the version and all other information for all the firmware devices in one request, use the `$expand` query parameter on the `FirmwareInventory` resource.

```
~ # curl
'http://192.168.1.2/redfish/v1/UpdateService/FirmwareInventory?$expand
=*'
{
      "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory",
```

```
        "@odata.type":
        "#SoftwareInventoryCollection.SoftwareInventoryCollection",
        "Members": [
        {
              "@odata.id":
              "/redfish/v1/UpdateService/FirmwareInventory/DEVICE_1",
              "@odata.type":
              "#SoftwareInventory.v1_4_0.SoftwareInventory",
              "Description": " DEVICE_1 image",
              "Id": "DEVICE_1",
              "Name": "Software Inventory",
              "RelatedItem": [
              {
                    "@odata.id": "/redfish/v1/Chassis/DEVICE_1"
              }
              ],
              "RelatedItem@odata.count": 1,
              "SoftwareId": "0x1000",
              "Status": {
              "Health": "OK",
              "HealthRollup": "OK",
              "State": "Enabled"
        },
        "Updateable": true,
        "Version": "XXX-22.10-1-123",
        "WriteProtected": false
},
……
{
        "@odata.id":
        "/redfish/v1/UpdateService/FirmwareInventory/DEVICE_2",
        "@odata.type": "#SoftwareInventory.v1_4_0.SoftwareInventory",
        "Description": "Other image",
        "Id": " DEVICE_2",
        "Manufacturer": "VENDOR","Name": "Software Inventory",
```

```
        "RelatedItem": [
        {
              "@odata.id": "/redfish/v1/Chassis/DEVICE_2"
        }
              ],
        "RelatedItem@odata.count": 1,
        "SoftwareId": "0x1000",
        "Status": {
              "Health": "OK",
              "HealthRollup": "OK",
              "State": "Enabled"
        },
        "Updateable": false,
        "Version": ""
        }
        ],
        "Members@odata.count": N,
        "Name": "Software Inventory Collection"
}
```

Appendix B

An open source tool has been developed as part of the compliance tool to pack and unpack a PLDM Type 5 FW package. It is located in https://github.com/microsoft/pldm-fw-pkgr/tree/main

It may be used by vendors and hyperscalers.