SPECIFICATION: FIRMWARE UPDATE

REQUIREMENTS FOR GPU

Author (s) :

Vishal Jain, Shivi Fotedar, Ryan Speiser ***NVIDIA***

Karunakara Kotary, Venkatesh Ramamurthy ***Microsoft***

Sujoy Sen, Arvind Ayyangar ***Google***

Executive Summary

Management of GPUs is not standardized, resulting in significant effort and time to onboard each new HW design. The lack of standardization is also a burden on suppliers who must accommodate varying requirements from their customers. This document describes industry standard formats and protocols for GPU System Firmware Update that make it easier for CSPs (Cloud Service Providers) to onboard new GPU and accelerator designs with less toil and faster time to market.

Table of Contents

1. Introduction **4**
1.1 Overview **4**
1.2 Assumptions, Constraints,Dependencies 4
1.3 Use Cases 5
1.4 Glossary/Terminology 5
2. Firmware Update Model **8**
2.1 Firmware Update States 9
2.2 Baseboard with Accelerator Management Controller (AMC) 9
2.3 Baseboard without AMC 11
2.4 Firmware Device State Machine Diagram 14
3. Firmware Update Requirements **16**
3.1 Firmware Update 16
3.1.1 Secure firmware update 17
3.1.2 Firmware Copy time 17
3.1.3 Firmware activation 17
3.1.4 Firmware Update protocols 17
3.1.5 Firmware update idempotency 18
3.1.6 Firmware update dependencies 18
3.1.7 Firmware date compatibility 18
3.1.8 Non-disruptive updates 18
3.1.9 Firmware update synchronization 18
3.1.10 Restore previous image upon cancellation of firmware update 19
3.2 Firmware Rollback Prevention 19
3.2.1 Checks to stop firmware rollback 19
3.2.2 Explicit command to update security versions on device 19
3.3 Firmware Image Format 19
3.4 Firmware Image Size 20
3.5 Firmware Fungibility 20
3.6 Single shot firmware update 21
3.7 Firmware Image redundancy 21
3.8 Firmware resiliency 21
3.8.1 Single Fault 21
3.8.2 Double Fault 22
3.9 Firmware telemetry 22
3.10 Debug messages for runtime DC Ops 22

4. Redfish Update Service **22**
4.1 Common properties 22
4.2 Firmware Inventory and Update Service 23
4.2.1 GET - UpdateService 23
4.2.2 GET - SoftwareInventory 26
4.2.3 GET - FirmwareInventory 27
4.2.4 POST - Updating Firmware 27
4.3 TaskService 29
4.3.1 Task 29
4.4 Redfish Error Reporting for Firmware Update 32
4.4.1 Message Registry 32
4.4.2 Immediate Failures 34
4.4.3 Failure during background/async operation 35
4.4.4 Firmware update client workflows 36
5. PLDM Update **40**
5.1. PLDM Update Commands 40
5.2 PLDM Update Timing Requirements 42
6. File Format **44**
6.1 File Format 44
6.2 Package Security 45
7. License - Open Web Foundation (OWF) CLA **46**
8. OCP Tenets **47**
About Open Compute Foundation **48**
Appendix A. [Redfish API Example] **49**

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 via openBMC.

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 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.
Background Update/Copy	This is the operation of sending a FW Image or Bundle to a GPU device or subsystem. This may consist of staging and/or flashing phases. The reason for the term “Background” is to imply that this does not disrupt the normal operation of the device during this process. The end state of this operation is the successful flashing of the FW Image.
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.
Fungibility	The property of a product whose individual units are interchangeable despite implementation differences. Fungible units must adhere to the same product definition in terms of form, fit, and compute value.

Firmware Fungibility	The ability for a single product firmware image or firmware bundle to support fungible (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.
Staging	This is non-volatile storage that a FW image may be temporarily stored in by a management controller during a FW Update process

Table 2: 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.
EC_FW	Embedded Controller Firmware	

Term	Elaboration	Description
	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 device and its flash storage.
FLR	Function level reset	Function level reset as defined by PCIE specifications.
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, and related specification).

Term	Elaboration	Description
SMC	Satellite 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.
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 3: Firmware Device Terminology

2. Firmware Update Model

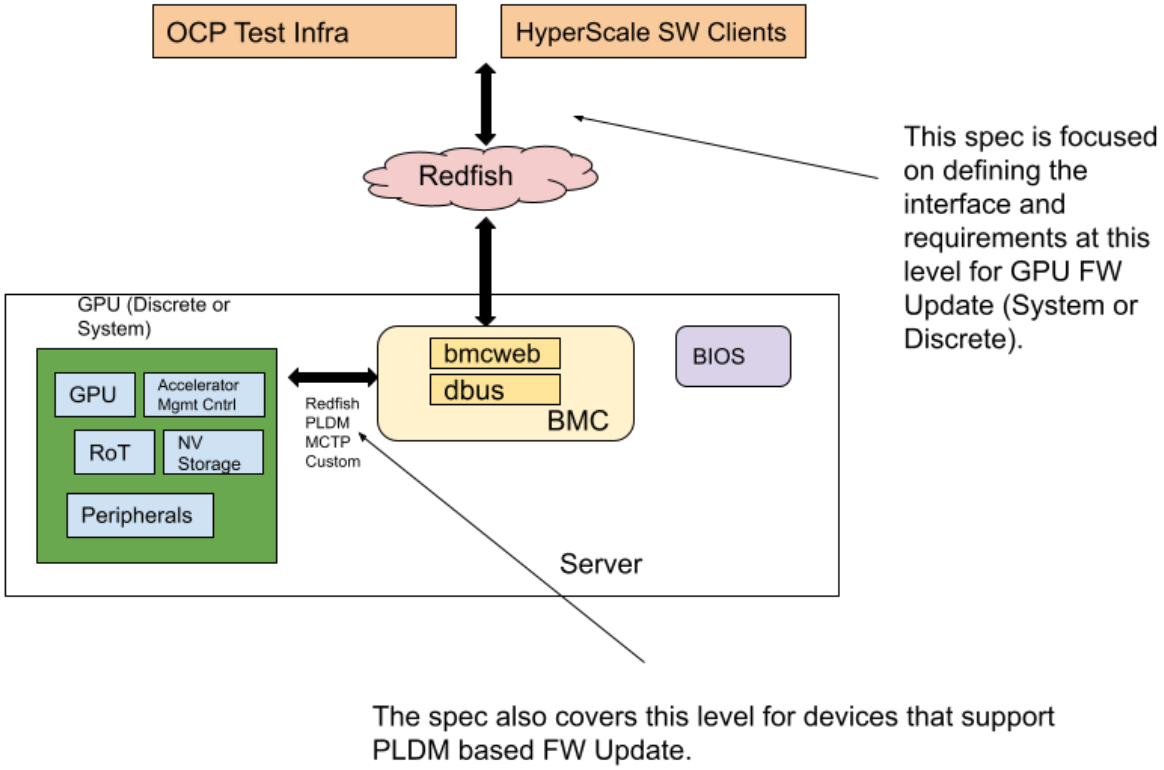


Figure 1: Hyperscaler Firmware update model

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 discrete GPUs, the management protocol between the BMC and the GPU is expected to be PLDM over MCTP.

2.1 Firmware Update States

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 sub-sections 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 doesn't have a AMC.

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.

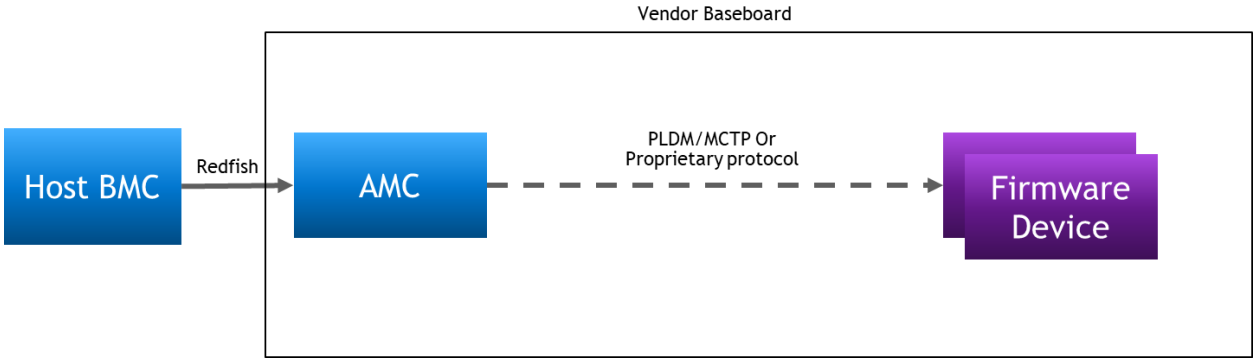


Figure 2: GPU Firmware device with AMC

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). SMC 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 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 SMC (via the host BMC).
2. The Orchestrator then uses Redfish UpdateService to copy ~~download~~ the firmware bundle to the AMC.
3. 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.
4. 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.
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 AMC notifies the Orchestrator (via the host BMC).
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 AMC.
10. SMC then relays the command to all the firmware devices, which then *Arm* their new images.
11. Once the Orchestrator reboots/power cycle/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.

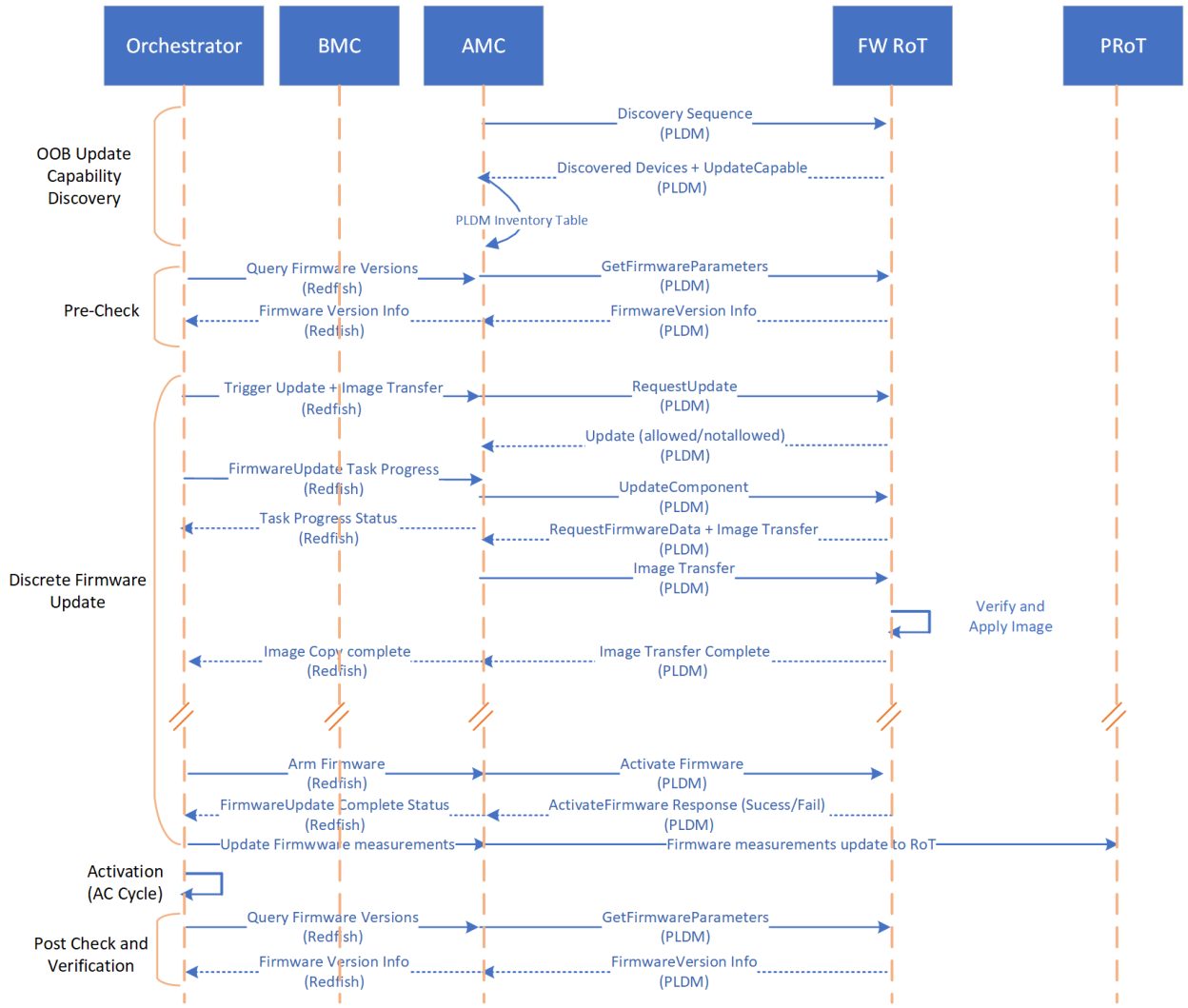


Figure 3: Firmware update sequence diagram with AMC

2.3 Baseboard without AMC

The figure below shows the host BMC directly updating firmware of all devices on a vendor board, when a vendor board doesn't have a 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.

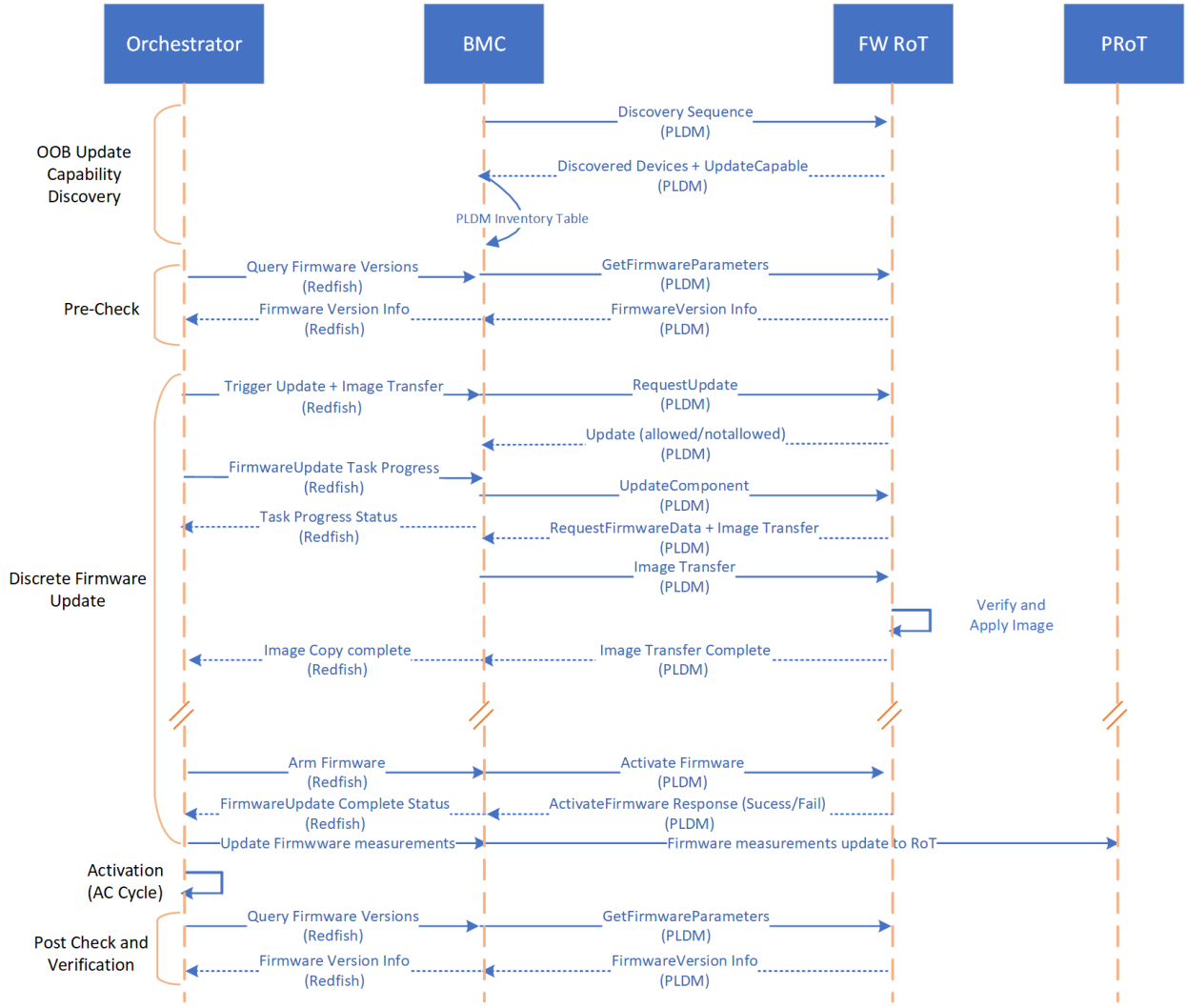


Figure 5: Firmware update sequence diagram without AMC

2.4 Firmware Device State Machine Diagram

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

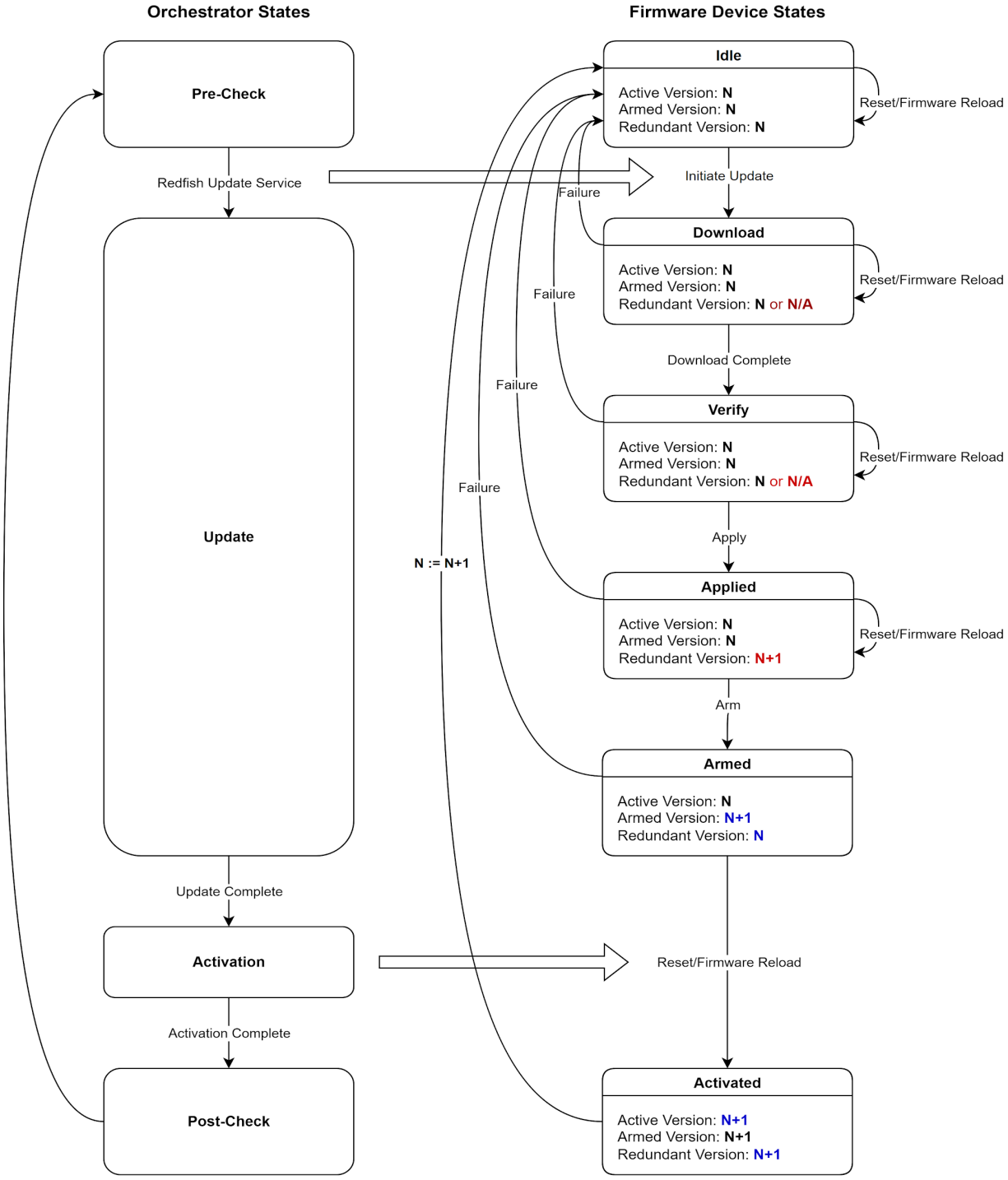


Figure 6: Firmware update device states

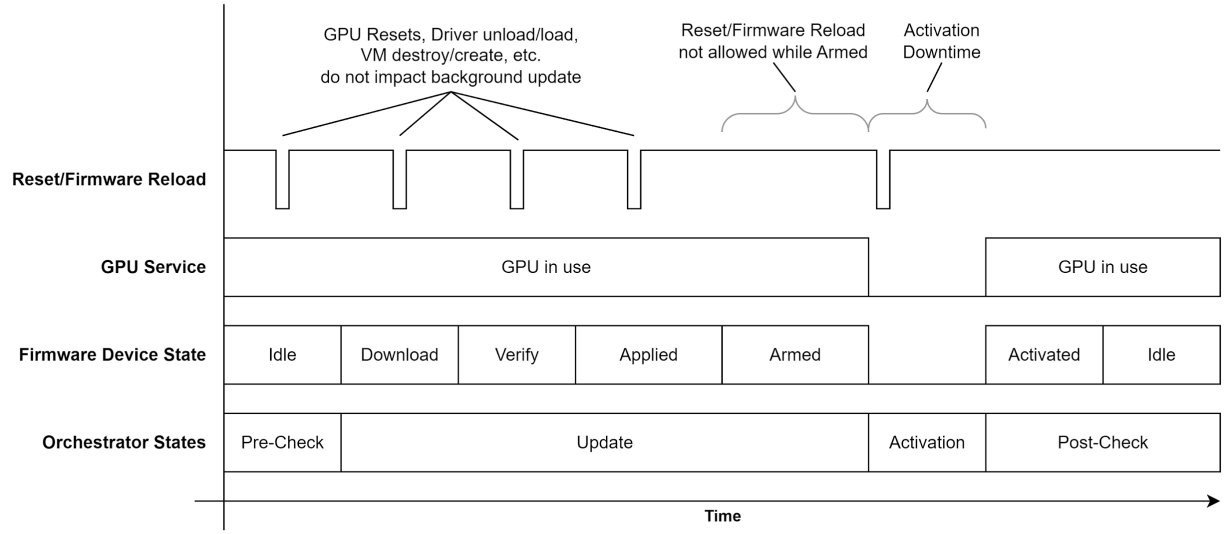


Figure 7: Firmware Device background update

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 redfish commands
Download	After receiving the appropriate Redfish command to update a firmware component, the FD moves to this state and works with Hyper scalar Update Agent over Redfish API to receive the complete firmware package & verifies that the PLDM package is intact by checking the checksum.
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. Gets called as part of Apply and as part of activation.

Apply	Upon completion of download, FD unwraps the PLDM package and verifies the UA redfish configured FD update targets, extracts the right firmware payloads & subjects them to verification. Upon successful verification step FD writes the firmware payload to the Nonvolatile storage of the respective FD.
Arm	Upon successful image application , Hyper scalar Update Agent may arm the image activation by configuring the appropriate activation method
Activate	Upon successful Application firmware images, Hyper scalar update agent requests FD to perform image activation , upon successful activation FD starts consuming the new firmware. After activation the FD moves to the IDLE state.

Table 4: 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	
Arm	Activate	Redfish/inband activate /AC/DC command	

Table 5: 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 Firmware update idempotency

Vendors shall support the option to force FW update even if FW versions match to the one in the update package. By default, devices shall perform FW version check and skip update if version matches to the one from the package.

3.1.6 Firmware update dependencies

There shall be no dependency between FW of different devices in a product. 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 4.2. 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/rollback 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. Following two sections address requirements to achieve this capability.

3.2.1 Checks to stop firmware rollback

To prevent usage of old (and possibly compromised) FW, a field needs to be supported for both SOC EFUSEs and FW binary image. This new field will be referred to in this document as 'FW Security Version' (SVN). Upon update and upon boot, the device FW or HW (depends on the device) will derive the FW security version from the EFUSEs and compare it to the SVN field in the FW binary image. In case the FW image SVN is not equal to or greater than the EFUSEs derived value, the device must indicate relevant error and stop the update.

3.2.2 Explicit command to update security versions on device

Devices shall support capability to update the security version 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.

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 Fungibility

A vendor may fulfill orders of a particular product with physically different, yet fungible (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 fungibility such that the same firmware image or firmware bundle provided for the product must support all fungible 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 fungible, units is outside the scope of this specification.

The requirement for a firmware image or firmware bundle to support all fungible 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 fungible 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 fungible HW variations not known at the time of firmware release. However, the firmware image or firmware bundle is required to support all previous fungible HW variations known at the time of firmware release, including those that have been discontinued.

When different firmware is required for fungible 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 fungibility 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 fungible units
- The firmware update package must have multiple Firmware Device Records, one for each of the uniquely identifiable fungible unit variations

- The Component Image information for the Firmware Device Record must describe and point to the corresponding firmware image for that fungible unit variation

Standard PLDM Type 5 device descriptors, such as PCI Device ID, may necessarily be the same between fungible 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 shall also provide the option to update one or more firmware in the firmware bundle as selected by the user.

3.7 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.

3.8 Firmware resiliency

3.8.1 Single Fault

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.

3.8.2 Double Fault

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 ~~device shall have a recovery mechanism to copy the current image and boot itself.~~ BMC or AMC should be able to copy current or some working image to recover the device. This recovery mechanism is required since the device will not failover to the inactive image as the device is not armed yet.

3.9 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

3.10 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 For UpdateService, it shall be “/redfish/v1/UpdateService”
@odata.type	string	Read-only	The type of a resource. For UpdateService, , it shall be “#UpdateService.v1_11_0.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. This value is unique within a collection 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 For UpdateService, it shall be “Update Service”

Table 6: GPU FW update Redfish API properties

4.2 Firmware Inventory and Update Service

4.2.1 GET - UpdateService

A GET on the 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.

URI: GET /redfish/v1/UpdateService

Input parameters: None

Schema Version: UpdateService.v1_11_0.json

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
HttpPushUriOptions	Object		The options for HttpPushUri provided software updates	Required
HttpPushUriOptionsBusy	Boolean	Read-write	An indication of whether a client has reserved the HttpPushUriOptions properties for software updates	Optional
HttpPushUriTargets	Array	Read-write	An array of URIs that indicate where to apply the update image	Optional
HttpPushUriTargetsBusy	Boolean	Read-write	An indication of whether any client has reserved the HttpPushUriTargets property	

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	Required
RemoteServerCertificates	Object		The link to a collection of server certificates of the server referenced by the ImageURI property in SimpleUpdate.	Optional
ServiceEnabled	Boolean	Read-write	An indication of whether this service is enabled	Required
SoftwareInventory	Object		An inventory of software	Required
VerifyRemoteServerCertificate	Boolean	Read-write	An indication of whether the service will verify the certificate of the server referenced by the ImageURI property in SimpleUpdate prior to sending the transfer request	Optional

Table 7: Redfish get UpdateService property

Example:

```
{
```

```
"@odata.type": "#UpdateService.v1_11_0.UpdateService",
"@odata.id": "/redfish/v1/UpdateService",
"Id": "UpdateService",
"Name": "Update Service",
"HttpPushUri": "<vendor-specific URI for push update>",
               "HttpPushUriTargets": [],
               "HttpPushUriOptions": {
                  "ForceUpdate": false,
                  "HttpPushUriApplyTime": {
                         "ApplyTime": "OnReset"
                         }
                   },
                   "MaxImageSizeBytes": 209715200,
                   "ServiceEnabled": true,
                   "FirmwareInventory": {
                         "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory"
                   },
                   "SoftwareInventory": {
                         "@odata.id": "/redfish/v1/UpdateService/SoftwareInventory"
                   },
}
```

4.2.2 GET - SoftwareInventory

The SoftwareInventory property is a resource collection containing the set of software components generally like drivers execute within a host operating system. Software in this collection is generally updated using platform-specific methods or utilities

```
{
    "@odata.type":
    "#SoftwareInventoryCollection.SoftwareInventoryCollection",
    "@odata.id": "/redfish/v1/UpdateService/SoftwareInventory",
    "Name": "Software Inventory Collection",
    "Members@odata.count": 1,
    "Members": [
        {
             "@odata.id":
           "/redfish/v1/UpdateService/SoftwareInventory/GPUDriver "
        }
    ]
}
```

4.2.3 GET - FirmwareInventory

The FirmwareInventory property is a resource collection containing 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

```
{
  "@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": "1",
  "Manufacturer": "Vendor",
  "Updateable": true,
  "WriteProtected": false,
  "Version": "1.0.0",
  "RelatedItem": [
    {
      "@odata.id": "/redfish/v1/Chassis/ ManagementController0"
    }

   ]

}
```

4.2.4 POST - Updating Firmware

For each updateable component the SimpleUpdate action can be invoked to update the firmware.

Actions / Description	**Parameter**	**Required/** **Optional**

StartUpdate – This action starts the firmware update	None		Optional
SimpleUpdate - This action updates software components	ImageURI	The URI of the software image to install	Required
	Targets	An array of URIs that indicate where to apply the update image	
	TransferProtocol	The network protocol that the update service uses to retrieve the software image file located at the URI provided in ImageURI. This parameter is ignored if the URI provided in ImageURI contains a scheme.	Optional
	Username	The user name to access the URI specified by the ImageURI parameter	Optional
	Password	The password to access the URI specified by the ImageURI parameter.	Optional

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.	Optional
HttpPusUriOptions	Object		The options for HttpPushUri provided software updates	Optional
HttpPushUriOptionsBusy	Boolean	Read-write	An indication of whether a client has reserved the HttpPushUriOptions properties for software updates	Optional

Table 8: Redfish POST UpdateService

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 9: 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.

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 indicated the supported Message Registry “#MessageRegistry.v1_4_1.MessageRegistry”	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 support qualify the message	Optional
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 10: 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 would respond back with TaskState, TaskStatus and Error with Message (as defined in Message Registry)
Device shall support the following immediate failure modes

Failure mode	**HTTP error code**	**MessageID**	**Other details**
Another Firmware Update already in progress	400	Update.1.0.UpdateInProgress	Message:”Another firmware update already in progress” MessageArgs: null Resolution: “Wait for the previous operation to complete before attempting another one”
Internal Error	500	Base.1.8.1.InternalError	Message:”The request failed due to an internal service error. The service is still operational” MessageArgs: null Resolution:”Resubmit the request. If the problem persists, consider resetting the service”
Incoming file is not PLDM file format	500	Base1.8.1.ResourceAtUriInUnknownFormat	Message:”The resource at <file/path> is in a format not recognized by the service” MessageArgs: “<filename of the uploaded file>” Resolution:”Upload a file supported by this device”
Insufficient Storage Space	500	Base.1.8.1.InsufficientStorage	Message:”” MessageArgs: Resolution:”Reset the baseboard and retry the operation.",”
Incoming file exceeds max device supported	500	Base.1.8.1.PayloadTooLarge	Message:”Requested file size exceeds the maximum supported for this <service>.” MessageArgs: “"/redfish/v1/UpdateService/"” Resolution:”Firmware package is

			greater than allowed size. Check the file before continuing”

Table 11: Redfish Immediate Failure codes

4.4.3 Failure during background/async operation

Once the device accepts the Update Service Firmware update command successfully, it returns a Task ID. Orchestrator keeps monitoring the task for completion. Completion can results 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).

Device shall support the following immediate failure modes

Failure mode	**HTTP error code**	**MessageID**	**Other details**
NVRAM ownership failed (SPI, ..)	200	ResourceEvent.1.1.0.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”
PLDM meta-data corrupt			Message:”” MessageArgs: Resolution:
Image has lower security version			Message:”” MessageArgs: Resolution:
Key Revocation failure			Message:”” MessageArgs: Resolution:
SKU Mismatch			Message:”” MessageArgs: Resolution:

Table 12: Background Failure codes

4.4.4 Firmware update client workflows

The workflow below shows how the API's discussed in section 4 work together to orchestrate FW update from clients perspective. This workflow is generic in the sense that it deals with HTTP response codes, Redfish Task, timeouts/retries and Redfish messages.

Following workflow orchestrates different states to facilitate "one-shot" FW update where a single URI combines Copy and ARM in one step from the client's perspective.

1. Start at **IDLE** state.
2. Check whether there is a current firmware update.
   This can happen when an error occurs on the client side, and it was reset after triggering the firmware update.

```
Get /redfish/v1/TaskService/Tasks

If Task exists such that TaskState == Running AND
Update.1.0.TargetDetermined in Task Messages array
    Move to UPDATE state, step 4 (this means a FW update is in progress)

Else If Task exists such that TaskState == Completed AND
Update.1.0.TargetDetermined in Messages array
    Exit at IDLE state (a FW update has completed and has to be activated)

Else
    Set RETRY_COUNT = 0
```

3. Provision the client with the firmware bundle.
4. Transfer the image to AMC.
   a. Start a five-minute timer, where the expiry handler is `XFER_TIMER_HANDLER`.
   b. Transfer the image by using `POST -T <firmware bundle> https://${amc}/redfish/v1/UpdateService`.

```
If  HTTP_RESONSE == 202 AND
    HAS_LOCATION_HEADER AND
    TaskState == Running AND
    TaskStatus == OK
        Reset RETRY_COUNT = 0
        Cancel timer from 3.a
        Move to step 4
Else
    If RETRY_COUNT == 3
        Exit to FAIL State
    Else
```

```
        Inc RETRY_COUNT
        Restart step 3 after 5 minutes
```

XFER_TIMER_HANDLER, which handles no response over Redfish:

```
If RETRY_COUNT == 3
    Exit to FAIL State
Else
    Inc RETRY_COUNT
    Restart step 3 after 5 minutes
```

5. Monitor the firmware update progress.
   a. Start a five-minute timer, and the handler is UPDATE_TIMER_HANDLER.
   b. Every five minutes, get the Redfish firmware update Task progress: GET /redfish/v1/TaskService/Tasks/{TaskId}

```
If  TaskState == Completed AND
    TaskStatus == OK AND
    PercentComplete == 100
        Stop Timer from 4.a
        Update Completed - exit to IDLE state
Else If  TaskState == Running AND
         TaskStatus == OK AND
         PercentComplete == <Changed from last poll>
             Cancel Timer from 4.a
Else
    If RETRY_COUNT == 3
       Report Task Messages array
       Exit to FAIL State
    Else
       Inc RETRY_COUNT
      Restart step 3 after 5 minutes
```

UPDATE_TIMER_HANDLER, which handles no response over Redfish:

```
If RETRY_COUNT == 3
    Exit to FAIL State
Else
    Inc RETRY_COUNT
    Restart step 3 after 5 minutes
```

Following workflow orchestrates different states to facilitate “split Copy/Arm” FW update

1. Start at INIT state.

Set RETRY_COUNT = 0

2. Provision the client with the firmware bundle, which moves the firmware update client to the **COPY** state.

3. Copy/Stage firmware bundle in HMC

This state can be achieved by two types of Redfish API's

- Unstructured HTTP push update
- Multi-Part HTTP push update

a. Start a five-minute timer, where the expiry handler is `XFER_TIMER_HANDLER`.

b. Transfer the firmware bundle by using Unstructured HTTP push update

i. `PATCH {"HttpPushUriOptions": { "HttpPushUriApplyTime": { "ApplyTime": "OnStartUpdateRequest"}}}`

ii. `POST -T <firmware bundle> https://${amc}/redfish/v1/UpdateService/update`

Transfer the firmware bundle by using Multi-part HTTP push update

i. `POST https://${amc}/redfish/v1/UpdateService/update-multipart`

```
Content-Type: multipart/form-data;
boundary=--------------------------d74496d66958873e
Content-Length: <computed-length>...
----------------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateParameters"
Content-Type: application/json
{
   "@Redfish.OperationApplyTime": "OnStartUpdateRequest",
}
Content-Disposition: form-data; name="UpdateFile";
filename="<firmware bundle>"
Content-Type: application/octet-stream

<firmware bundle binary>
```

```
IF  HTTP_RESONSE == 201 AND
    HAS_LOCATION_HEADER

    IF RETRY_COUNT == 3
          Exit to FAIL State
    ELSE
          Inc RETRY_COUNT
          Restart step 3 after 5 minutes
```

`XFER_TIMER_HANDLER`, which handles no response over Redfish:

```
If RETRY_COUNT == 3
```

```
    Exit to FAIL State
ELSE
    Inc RETRY_COUNT
    Restart step 3 after 5 minutes
```

4. Arm the copied firmware to be ready for update
    a. Start a five-minute timer, where the expiry handler is `START_UPDATE_TIMER_HANDLER`.
    b. Initiate firmware update by using `POST /redfish/v1/UpdateService/Actions/UpdateService.StartUpdate`

```
IF  HTTP_RESONSE == 202 AND
    HAS_LOCATION_HEADER AND
    TaskState == Running AND
    TaskStatus == OK
        Reset RETRY_COUNT = 0
        Cancel timer from step 4.a
        Move to step 5
ELSE
    IF RETRY_COUNT == 3
        Exit to FAIL State
    ELSE
        Inc RETRY_COUNT
        Restart step 4 after 5 minutes
```

`START_UPDATE_TIMER_HANDLER`, which handles no response over Redfish:

```
If RETRY_COUNT == 3
    Exit to FAIL State
ELSE
    Inc RETRY_COUNT
    Restart step 4 after 5 minutes
```

5. Monitor the firmware update progress. It should move to the **ARM** state
    a. Start a five-minute timer, and the handler is `UPDATE_TIMER_HANDLER`.
    b. Every five minutes, get the Redfish firmware update Task progress: `GET /redfish/v1/TaskService/Tasks/{TaskId}`

```
If  TaskState == Completed AND
    TaskStatus == OK AND
    PercentComplete == 100
        Stop Timer from step 5.a
        Update Completed - exit to IDLE state
ELSE IF  TaskState == Running AND
```

```
        TaskStatus == OK AND
        PercentComplete == <Changed from last poll>
            Cancel Timer from step 5.a
ELSE
    IF RETRY_COUNT == 3
       Report Task Messages array
       Exit to FAIL State
    ELSE
       Inc RETRY_COUNT
      Restart step 4 after 5 minutes
```

`UPDATE_TIMER_HANDLER`, which handles no response over Redfish:

```
If RETRY_COUNT == 3
    Exit to FAIL State
ELSE
    Inc RETRY_COUNT
    Restart step 4 after 5 minutes
```

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 13: 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 v1.2.0 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 v1.2.0 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	2		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.

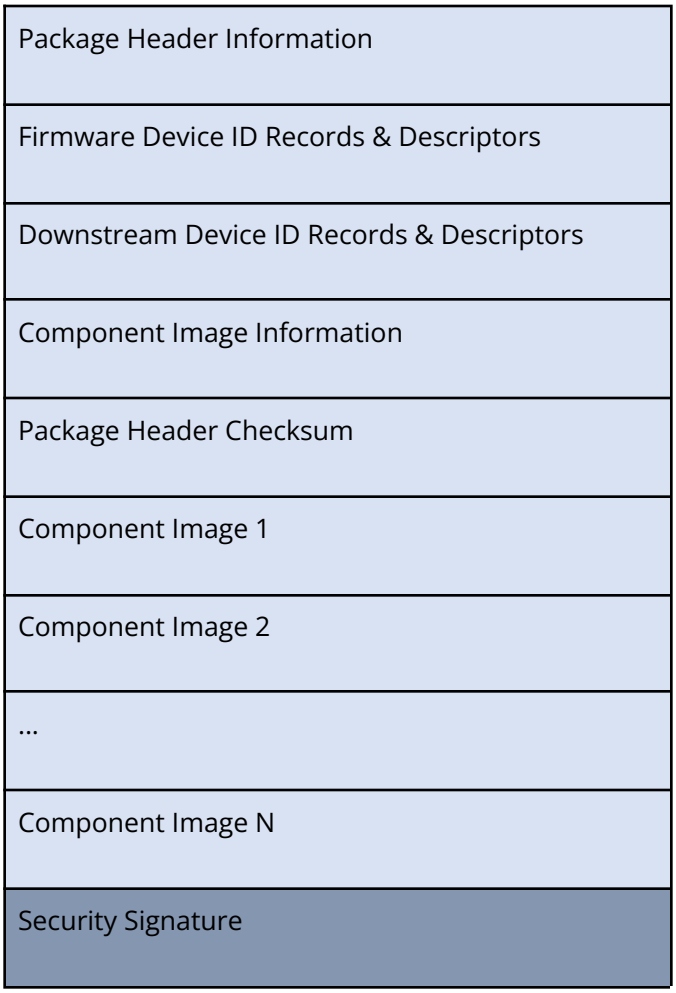
Request firmware data idle timeout	60s	90s	Amount of time for the Update Agent to cancel the component update if no command is received from the FD/FDP when it's expected, during the component image transfer stage.
State change timeout	180s		Amount of time for the Update Agent to wait before canceling 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 14: PLDM Update timing requirements

6. File Format

Supported file format for Redfish Update Service is DMTF PLDM File format as described in DMTF DSP0267 v1.2.

6.1 File Format



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. License - Open Web Foundation (OWF) CLA

Contributions to this Specification are made under the terms and conditions set forth in Open Web Foundation Modified Contributor License Agreement ("OWF CLA 1.0") ("Contribution License") by:

Google, Microsoft, NVIDIA

Usage of this Specification is governed by the terms and conditions set forth in **Open Web Foundation Modified Final Specification Agreement ("OWFa 1.0.2") ("Specification License").**

You can review the applicable OWFa1.0 Specification License(s) referenced above by the contributors to this Specification on the OCP website at http://www.opencompute.org/participate/legal-documents/.For actual executed copies of either agreement, please contact OCP directly.

Notes:

1. The above license does not apply to the Appendix or Appendices. The information in the Appendix or Appendices is for reference only and non-normative in nature.

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8. OCP Tenets

Openness

- This specification was developed via close and open collaboration between industry partners and competitors.
- All specifications and interfaces produced through this effort will be available to all OCP members.

Efficiency

- The goal of this specification is to make integration of GPUs into Hypercaler solutions seamless, reducing the toil for both the supplier and the Hyperscaler consumers.
- A companion to this effort is the OCP compliance tool that will enable automated validation of these interfaces, ensuring reduced toil and high-quality products.

Impact

- This document represents the first industry initiative to standardize GPU requirements between suppliers and hyperscale consumers.
- The advances in this document are expected to have significant impact on quality and time-to-market for GPU systems deployed by hyperscalers.
- These advances will also be applicable and beneficial to enterprise deployments of GPU systems.

Scale

- This specification applies to very large scale GPU system deployments in Hyperscale Data Centers

Sustainability

- The profiles defined in this document enable cross-generational commonality for key functionality of GPU parts, enabling logistics to support longer lifespan of GPU parts and a healthy secondary market for these parts.

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]

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 POSTed is in the HttpPushUri property.
curl http://${smc}/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 SMC using HTTP POST:
curl -X POST -T <firmware update package>
http://${smc}/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 URIs 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://${smc}/redfish/v1/UpdateService

b. Repeat step 1b.

Note: The firmware update package needs to be pushed (HTTP POSTed) 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 (step 1b) is an asynchronous operation (see DSP0266). The SMC's Redfish service will implement the TaskService schema and return a Redfish Task as the
response to the HTTP post and a 202 Accepted HTTP response code:

```
< HTTP/1.1 202 Accepted
< Location: /redfish/v1/TaskService/Tasks/7/Monitor
< Retry-After: 30
< Content-Type: application/json
<
{
"@odata.id": "/redfish/v1/TaskService/Tasks/7",
"@odata.type": "#Task.v1_4_3.Task",
"Id": "7",
"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 SMC 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

```
< HTTP/1.1 200 OK
< Content-Type: application/json
{
"@odata.id": "/redfish/v1/TaskService/Tasks/7",
"@odata.type": "#Task.v1_4_3.Task",
"Id": "7",
"Messages": [
{
"@odata.type": "#Message.v1_0_0.Message",
```

"Message": "The task with id 7 has started.",
"MessageArgs": [
"7"
],
"MessageId": "TaskEvent.1.0.1.TaskStarted",
"Resolution": "None.",
"Severity": "OK"
},
{
"@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
"Message": "The target device 'FW_SMC_0' will be updated
with image 'cec1736ApFw-12345678'.",
"MessageArgs": [
"FW_SMC_0",
"cec1736ApFw-12345678"
],
"MessageId": "Update.1.0.TargetDetermined",
"Resolution": "None.",
"Severity": "OK"
},
{
"@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
"Message": "Image 'cec1736ApFw-12345678' is being transferred to
'FW_SMC_0'.",
"MessageArgs": [
"cec1736ApFw-12345678",
"FW_SMC_0"
],
"MessageId": "Update.1.0.TransferringToComponent",
"Resolution": "None.",
"Severity": "OK"
}
],
"Name": "Task 7",
"Payload": {
"HttpHeaders": [
"Host: 192.168.31.1",
"User-Agent: curl/7.64.0",
"Accept: */*",
"Content-Length: 67105992"
],
"HttpOperation": "POST",
"JsonBody": "null",
"TargetUri": "/redfish/v1/UpdateService"
},

```
"PercentComplete": 0,
"StartTime": "2023-03-16T02:18:38+00:00",
"TaskMonitor": "/redfish/v1/TaskService/Tasks/7/Monitor",
"TaskState": "Running",
"TaskStatus": "OK"
}
```


Task Resource Response: When the task is completed.

< HTTP/1.1 200 OK


```
{
"@odata.id": "/redfish/v1/TaskService/Tasks/7",
"@odata.type": "#Task.v1_4_3.Task",
"EndTime": "2023-03-16T02:27:31+00:00",
"Id": "7",
"Messages": [
{
"@odata.type": "#Message.v1_0_0.Message",
"Message": "The task with id 7 has started.",
"MessageArgs": [
"7"
],
"MessageId": "TaskEvent.1.0.1.TaskStarted",
"Resolution": "None.",
"Severity": "OK"
},{
"@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
"Message": "The target device 'FW_SMC_0' will be updated
with image 'cec1736ApFw-12345678'.",
"MessageArgs": [
"FW_SMC_0",
"cec1736ApFw-12345678"
],
"MessageId": "Update.1.0.TargetDetermined",
"Resolution": "None.",
"Severity": "OK"
},
{
"@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
"Message": "Image 'cec1736ApFw-12345678' is being transferred to
'FW_SMC_0'.",
"MessageArgs": [
"cec1736ApFw-12345678",
"FW_SMC_0"
],
"MessageId": "Update.1.0.TransferringToComponent",
"Resolution": "None.",
```

```
"Severity": "OK"
},
{
"@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
"Message": "Device 'FW_SMC_0' successfully updated with
image 'cec1736ApFw-12345678'.",
"MessageArgs": [
"FW_SMC_0",
"cec1736ApFw-12345678"
],
"MessageId": "Update.1.0.UpdateSuccessful",
"Resolution": "None.",
"Severity": "OK"
},
{
"@odata.type": "#MessageRegistry.v1_4_1.MessageRegistry",
"Message": "Awaiting for an action to proceed with activating
image 'cec1736ApFw-12345678' on 'FW_SMC_0'.",
"MessageArgs": [
"cec1736ApFw-12345678",
"FW_SMC_0"
],
"MessageId": "Update.1.0.AwaitToActivate",
"Resolution": "System reboot or AC power cycle",
"Severity": "OK"
},
{
"@odata.type": "#Message.v1_0_0.Message",
"Message": "The task with id 7 has Completed.",
"MessageArgs": [
"7"
],
"MessageId": "TaskEvent.1.0.1.TaskCompletedOK",
"Resolution": "None.",
"Severity": "OK"
}
],
"Name": "Task 7",
"PercentComplete": 100,
"StartTime": "2023-03-16T02:18:38+00:00",
"TaskState": "Completed",
"TaskStatus": "OK"
}
```


- Task Monitor URI Lifecycle

The Task monitor URI response contains a task ID that can be used to get firmware update status

when the task is running or completed. In the current implementation, the task monitor URI will disappear when the task is complete. Use the task resource to get the firmware update status after the task monitor URI stops responding with status code 202 and returns code 404.

Note: The task monitor URI does not have complete task representation, we recommend that you use the task resource URI to monitor the firmware update and not the task monitor URI.

Task Monitor response: When the task is running

```
< HTTP/1.1 202 Accepted
< Location: /redfish/v1/TaskService/Tasks/7/Monitor
< Retry-After: 30
< Content-Type: application/json
{
"@odata.id": "/redfish/v1/TaskService/Tasks/7",
"@odata.type": "#Task.v1_4_3.Task",
"Id": "7",
"TaskState": "Running",
"TaskStatus": "OK"
}
```

Task Monitor response: When the task is completed

```
< HTTP/1.1 404 Not Found
{
"error": {
"@Message.ExtendedInfo": [
{
"@odata.type": "#Message.v1_1_1.Message",
"Message": "The requested resource of type Monitor named '7' was
not found.",
"MessageArgs": [
"Monitor",
"7"
],
"MessageId": "Base.1.13.0.ResourceNotFound",
"MessageSeverity": "Critical",
"Resolution": "Provide a valid resource identifier and resubmit
the request."
}
],
"code": "Base.1.13.0.ResourceNotFound",
"message": "The requested resource of type Monitor named '7' was not
found."
}
}
```

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

```
curl
http://${smc}/redfish/v1/UpdateService/FirmwareInventory/DEVICE_1
{
"@odata.id":
"/redfish/v1/UpdateService/FirmwareInventory/ DEVICE_1",
"@odata.type": "#SoftwareInventory.v1_1_0.SoftwareInventory",
"Description": "Other image",
"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://192.168.31.1/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": " FW_DEV_0 image",
```

"Id": " FW_DEV_0",
"Name": "Software Inventory",
"RelatedItem": [
{
"@odata.id": "/redfish/v1/Chassis/ FW_DEV_0"
}
],
"RelatedItem@odata.count": 1,
"SoftwareId": "0x0010",
"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": "",
"Status": {
"Health": "OK",
"HealthRollup": "OK",
"State": "Enabled"
},
"Updateable": false,
"Version": ""
}
],
"Members@odata.count": N,
"Name": "Software Inventory Collection"
}