

OCP GPU & ACCELERATOR MANAGEMENT

INTERFACES

Version 1.1

Authors: (in alphabetical order)

David Blocker, NVIDIA

James Bodner, NVIDIA

Weifu Chen, Microsoft

Deepak Kodihalli, NVIDIA

Shrikant Giridhar, NVIDIA

Johan van de Groenendaal, NVIDIA

Vishal Jain, NVIDIA

Justin King, AMD

Choudary Maddukuri, Microsoft

Rajat Madhusudan, Microsoft

Hari Ramchandran, Microsoft

Sivakumar Sathappan, AMD

Krishna Sugumaran, AMD

Balaji Vembu, Meta

Linda Wu, NVIDIA

Justin York, 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 that make it easier for CSPs (Cloud Service Providers, hyperscalers) to onboard new GPU and accelerator designs with less toil and faster time to market while reducing manageability permutations for suppliers.

Table of Contents

1 INTRODUCTION 3

2 OVERVIEW 3

2.1 Discrete Accelerator Devices Manageability 3

2.1.1 Static discovery 4

2.1.2 Transport Protocol 4

2.1.3 Attestation 4

2.1.4 Management capability discovery 4

2.1.5 Base monitoring and control 4

2.1.6 Firmware management 5

2.2 UBB Accelerator Devices Manageability 5

2.2.1 Static discovery 5

2.2.2 Attestation 5

2.2.3 Base monitoring and control 6

2.2.4 Firmware management 6

3 DISCRETE ACCELERATOR DEVICE INTERFACES 6

3.1 Access Frequency 6

3.2 FRU Access and Data Model 6

3.3 MCTP 8

3.3.1 Bus Owner and EIDs 8

3.3.2 MCTP Discovery 8

3.3.3 MCTP command details 8

3.4 SPDM 10

3.5 PLDM 10

3.5.1 Discrete GPU/Accelerator Modules 10

3.5.2 PLDM Sensors & Effecters 11

3.5.2.1 Sensors 11
3.5.2.2 Effecters 12
3.5.3 Entity Association 12
3.5.3.1 Physical Entity Association 12
3.5.3.2 Logical Entity Association 13
3.5.4 Sensor & Effecter Entity Mapping 13
3.5.4.1 Numeric Sensors 13
3.5.4.2 State Sensors 13
3.5.4.3 Effecters 13
3.5.4.4 Logical Sensor Mapping 14
3.5.5 PLDM Type IDs 14

3.6 MCTP OEM VDM 14
3.6.1 Management Protocol Structures 14
3.6.1.1 Conventions 15
3.6.1.2 Data Type Enumeration 15
3.6.1.3 OCP VDM binding using PCI Vendor ID 16
3.6.1.3.1 Field Descriptions for OCP VDM binding using PCI Vendor ID 17
3.6.1.4 OCP VDM binding using OCP's IANA Enterprise ID 18
3.6.1.4.1 Field Descriptions for OCP VDM binding using OCP's IANA Enterprise ID 18
3.6.1.5 Message Payload Format 19
3.6.1.5.1 Discrete Request Message Format (OCP Version 1) 19
3.6.1.5.1.1 Field Descriptions 19
3.6.1.5.2 Discrete Request Message Format (OCP Version 2) 20
3.6.1.5.2.1 Field Descriptions 20
3.6.1.5.3 Discrete Response Message Format 20
3.6.1.5.4 Discrete Response message with data 21
3.6.1.5.4.1 Fields Descriptions 21
3.6.1.5.5 Discrete Response message with reason code 22
3.6.1.5.5.1 Fields Descriptions 22
3.6.1.5.6 Bulk Telemetry Request Message Format 22
3.6.1.5.6.1 Field Descriptions 22
3.6.1.5.7 Bulk Telemetry Response Message 23
3.6.1.5.7.1 Bulk Telemetry Fields Descriptions 23
3.6.1.5.7.2 Example 25

3.6.2 Completion Codes 26
3.6.3 Reason Codes 27
3.6.4 OEM Events 27
3.6.4.1 Fields Descriptions 27
3.6.4.2 OEM Event Classes 28
3.6.4.3 OEM Event Acknowledgement 29
3.6.4.3.1 Field Descriptions 30
3.6.5 Timing Specification 30

4 UBB ACCELERATOR DEVICE INTERFACES 31

4.1 Redfish 31
4.1.1 Redfish Tree 31
4.1.2 Location Objects 32
4.1.3 Redfish Interoperability Profile (RIP) 32
4.1.4 DMTF Redfish Message Registry 33

4.2 Out-of-Band Interfaces (MCTP, PLDM, and SPDM) 33

4.3 SPDM Over Redfish 33

5 CONCLUSION 35

6 GLOSSARY 35

7 REFERENCES 35

8 LICENSE - OPEN WEB FOUNDATION (OWF) CLA 36

9 OCP TENETS 37

10 ABOUT OPEN COMPUTE FOUNDATION 38

1 Introduction

Advances in accelerator-based workloads are driving the need for ever-faster bring-up and deployment of new accelerator-based system designs. Different accelerator components from different vendors implement a wide variety of management interfaces with little commonality. This lack of commonality causes extra toil and increases time-to-market for deploying new, innovative designs.

This document describes a common and generic framework for managing GPU and accelerators by way of management interfaces designed using standards-based management protocols.

This document covers both discrete accelerator devices (e.g., PCI-e CEM cards) and Universal Baseboard (UBB) designs (e.g., OCP OAI HGX).

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

2 Overview

The section specifies the interfaces for managing both Discrete and UUB design GPU/Accelerators.

2.1 Discrete Accelerator Devices Manageability

Discrete accelerator devices are GPUs packaged as standard CEM form-factor PCI-e adapters. Discrete accelerator device management can be broken down into several categories, each of which tie-in to various industry standards.

Table 1 – DMTF Standard Protocols

Manageability Objective	Technology	Standard
Static discovery	IPMI FRU	Platform Management FRU Information Storage Definition (rev. 1.3)
Transport protocol	MCTP	DMTF DSP0236 revision 1.3.1 or later
Attestation	SPDM	DMTF DSP0274 revision 1.2.1 or later
Management capability discovery	PLDM Type 0	DMTF DSP0240 revision 1.1.0 or later
Base monitoring and control	PLDM Type 2	DMTF DSP0248 revision 1.2.2 or later
Accelerator monitoring and control	PLDM Type 2	DMTF DSP2061 revision 1.0
Firmware management	PLDM Type 5	DMTF DSP0267 revision 1.2.0 or later

File I/O	PLDM Type (Future/TBD)	DMTF DSP0242
PLDM Set State	PLDM	DSP0249 revision 1.1.0

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

2.1.1 Static discovery

The initial, static discovery of devices seeks to obtain primarily immutable information about the discrete accelerator device. This is done by reading the values from a dedicated FRU component (e.g. EEPROM) contained within the device.

2.1.2 Transport Protocol

MCTP is the media independent protocol for communication among management controllers within the system. The corresponding section below describes the Bus owner relationship, the endpoint discovery, EID assignment, the bridges, and the EID pool assignments and corresponding flows.

2.1.3 Attestation

Once static and MCTP endpoint discovery is complete, an important step is attestation of the device. SPDM (DSP0274) operates over MCTP as a standardized mechanism to authenticate hardware identity and measure firmware identity. Using SPDM, the management controller serves as an SPDM requestor and discovers and negotiates the security capabilities to be used by the responder (discrete accelerator device). The management controller then authenticates the responder. Lastly, the management controller requests firmware measurements from the responder.

Note: SPDM attestation may happen repeatedly at the discretion of the management controller and is not limited to when the discrete accelerator device is initially discovered.

2.1.4 Management capability discovery

Once the hardware discrete accelerator device has been authenticated and measured, continuous management begins by discovering the management capabilities of the device. For standards-based management, subsequent discovery of the device capabilities happens by way of PLDM Type 0 (base) discovery.

2.1.5 Base monitoring and control

Once basic manageability capabilities are discovered dynamically from the device, continuous monitoring and control is performed by way of PLDM Type 2 (DSP0248).

2.1.6 Firmware management

Standards-based device firmware management requires support for PLDM Type 5 (DSP0267). DSP0267 provides a standardized way to query the version of the device's firmware (inventory) and to push firmware to the device (update).

2.2 UBB Accelerator Devices Manageability

Universal baseboard (UBB) designs are multi-GPU/accelerator devices with high-speed intra-GPU interconnects, PCI-e switches and retimers, and complex topologies. In those HW devices the expectation is the UBB must present a Redfish interface and may additionally support MCTP for high-frequency telemetry (e.g. thermal loop data) and attestation.

The Redfish interface may be exposed by accelerator management controller (AMC) on/or adjacent to the UBB or, in the future, via a software library (virtual AMC) provided by the supplier (details to defined in a future version of this document).

Table 2 – DMTF Standard Protocols

Manageability Objective	Technology	Description
Static discovery	IPMI FRU	Platform Management FRU Information Storage Definition (rev. 1.3)
Transport protocol	Redfish/MCTP	Redfish via NW and MCTP I2C/I3C/USB, etc...
Attestation	SPDM	Redfish or MCTP
Accelerator monitoring and control	Redfish/PLDM Type 2	Comprehensive monitoring via Redfish and High frequency telemetry via I2C/I3C/USB, etc..
Firmware management	Redfish	Via Redfish

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

2.2.1 Static discovery

UBB accelerator designs are expected to present a FRU EEPROM or virtual FRU equivalent (identical to what is required for discrete accelerator devices) and an AMC with a Redfish and optionally an MCTP interface w/ PLDM.

2.2.2 Attestation

SPDM attestation is managed by the hyperscaler BMC but the low-level attestation is through the AMC Redfish interface or through standard SPDM/MCTP.

2.2.3 Base monitoring and control

UBB accelerator designs are managed predominantly through Redfish implemented by the UBB AMC. High frequency and critical telemetry like power and thermal are managed via MCTP or PLDM over a sideband connection such as I2C/I3C/USB.

2.2.4 Firmware management

UBB accelerator designs shall support firmware update through Redfish implemented by the AMC.

3 Discrete Accelerator Device Interfaces

This section lists the minimum set of protocols and OEM extension/format that any GPU/accelerator vendor needs to support on their discrete accelerator devices for hyperscalers to seamlessly manage these devices without any additional vendor/device specific work.

3.1 Access Frequency

The following table provides a high-level mapping of Data/Telemetry categories to protocols including the required frequency of access that must be reliably supported by the accelerator device.

Table 3 – Discrete Accelerator Management Protocols

Telemetry Data/Attribute (unordered)	Protocol	Use Case	Frequency
Monitoring (Sensors)	PLDM Type 2 (DSP0248) and MCTP OCP OEM	e.g. Thermal, power, counters…	1s
Discovery	MCTP		On demand
Support Dump	Future direction: PLDM Type DSP0242	Comprehensive triage	On demand
Debug Logs	Future direction: PLDM Type (DSP0242		On demand
Updates	PLDM Type 5 (DSP0267)		
Configuration/Settings	PLDM Type 2 (Effecters) (DSP0248)		On demand

3.2 FRU Access and Data Model

Static discovery of the discrete accelerator device is accomplished by accessing an I2C/I3C FRU EEPROM or, if not present, through virtual FRU devices. In the case of a virtual FRU, it must support the following requirements:

- Must be available for access on AUX power
- Must be accompanied by recovery design in the event that the virtual FRU fails

In either case, the FRU device must return IPMI 1.3 FRU formatted data supporting the following requirements:

- Min 512bytes, ideally 4k (512byte min forces 2byte offsets)
- Internal Use Area: Optional
- Chassis Info Area: Optional
- Board Info Area: Mandatory (see below)
- Product info area: Mandatory (80 Bytes Minimum)
- Multi Record Area: Mandatory for any OEM extensions and the last area so that it extends.

It is mandatory that a FRU data source is only accessed by a single agent (e.g. BMC on Host Platform). Allowing multiple agents to access a FRU data source can lead to FRU data source data corruption and reduce system reliability.

The following table lists FRU data areas that must be populated.

Table 4 – IPMI FRU Data Fields

FRU Area	Fields	Required w/ Valid Value
Common Header	Common Header	Y
Chassis Info Area	Chassis Type	N
	Chassis Part Number	N
	Chassis Serial Number	N
Board Info Area	Mfg Date/Time	N
	Board Manufacturer	Y
	Board Product Name	Y
	Board Serial Number	Y
	Board Part Number	Y
Product Info Area	Manufacturer Name	Y
	Product Name	Y
	Product Part/Model Number	Y
	Product Version	N
	Product Serial Number	Y
Multi-Record Area	Mandatory for any OEM extensions and the last area so that it extends.	Y, if applicable

3.3 MCTP

This section describes the MCTP discovery and inventory flows and lists minimum required commands. This section also proposes and defines a high-level generic VDM command format for data that needs OEM extensions.

3.3.1 Bus Owner and EIDs

Typically, the hyperscaler's BMC is the topmost bus owner and all other devices are endpoint devices. The endpoint devices can themselves be a bus owner when they are connected to 2 or more MCTP buses (also referred to as a "bridged device"). The "topmost bus owner" is responsible for allocating all EIDs. It assigns a pool of EIDs to the bridge devices so that they allocate those EIDs to the devices behind them.

3.3.2 MCTP Discovery

MCTP discovery/re-discovery happens on the following conditions:

- AC power ON
- DC power cycle
- BMC reset
- Device reset

Following an AC/DC power ON, the discovery can occur any time during the system boot and typically complete before the host operating system has booted. The end device should be ready in less than 60sec from power ON and the PCI-e enumeration must not impact the I2C/I3C communication.

The flow charts below detail the MCTP discovery and EID assignment flows.

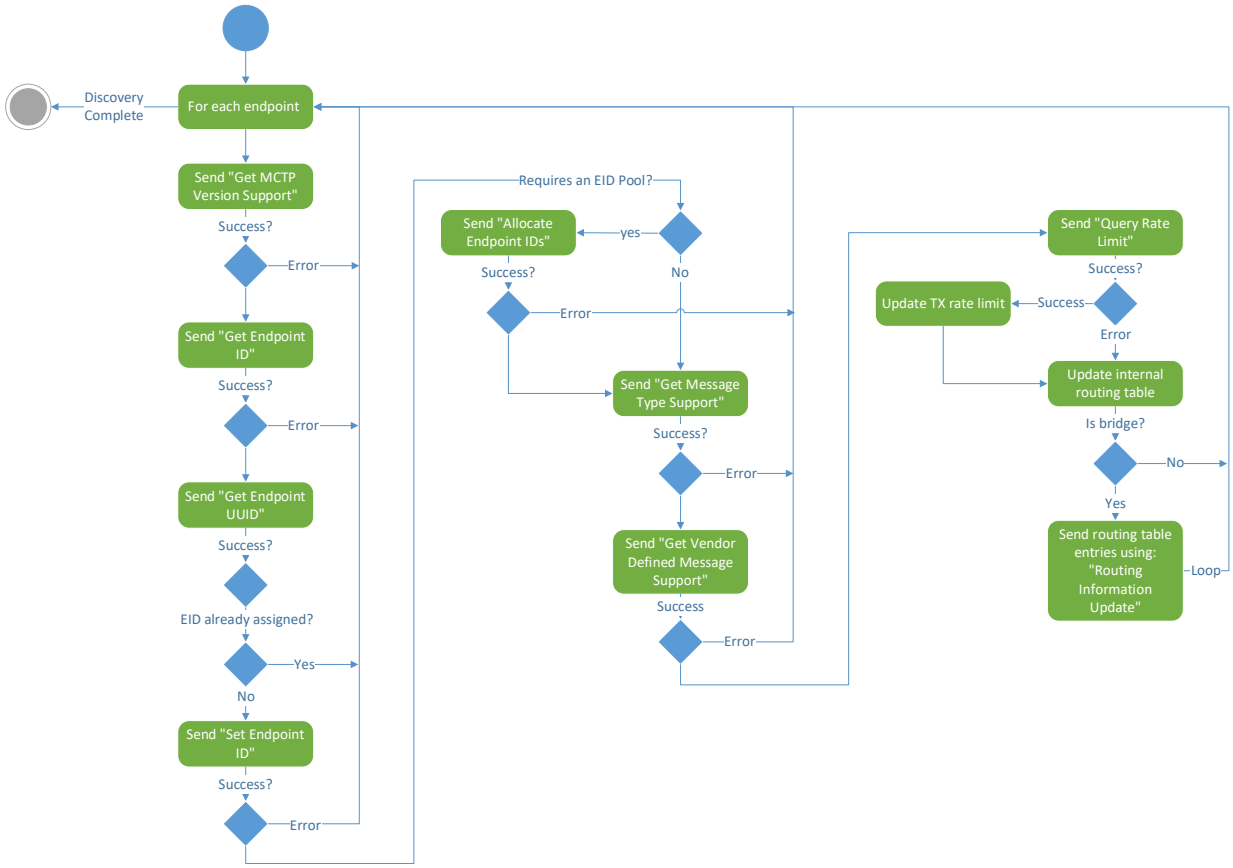


Figure 1 – MCTP Discovery Flow

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.3.3 MCTP command details

Table 5 lists the MCTP control commands and if they are required for use. This table is very similar to Table 12 within the DSP0236 specification with deviations indicated by superscripts.

Table 5 – MCTP Commands

Cmd Code	Command Name	Endpoint		Bridge		Notes
0x01	Set Endpoint ID	**Ma**	*Ng*	**Ca**[Ma]	**Mg**	
0x02	Get Endpoint ID	**Ma**	Og	**Ma**	**Og**[Mg]	
0x03	Get Endpoint UUID	**Ca**[1]	Og	**Ca**[1]	Og	1. Mandatory if device connects to multiple buses.

0x04	Get MCTP Version Support	**Ma**	Og	**Ma**	**Og**[Mg]	1. Bridges must use this command to verify compatible MCTP control protocol versions.
0x05	Get Message Type Support	**Ma**	Og	**Ma**	Og	
0x06	Get Vendor Defined Message Support	**Oa**[Ca1]	Og	**Oa**[Ca1]	Og	1. Mandatory to accept if device used vendor defined messaging.
0x07	Resolve Endpoint ID	*Na*	Og	**Ma**	Og	
0x08	Allocate Endpoint IDs	*Na*	Ng	**Ma**	*Mg*[Ng]	
0x09	Routing Information Update	Oa	Og	**Ma**	***Mg***[Ng]	
0x0A	Get Routing Table Entries	*Na*	Og	**Ma**	Og	
0x0B	Prepare for Endpoint Discovery	**Ca**[1]	*Ng*	**Ca**[1]	Cg[Ng]	1. Mandatory on a per bus basis to support endpoint discovery.
0x0C	Endpoint Discovery	**Ca**[1]	*Ng*	**Ca**[1]	Cg[Ng]	1. Mandatory on a per bus basis to support endpoint discovery.
0x0D	Discovery Notify	*Na*	**Cg**[1]	**Ca**[1]	**Cg**[1]	1. Mandatory if physical binding supports hot joins (e.g. I3C and PCI VDM).
0x0E	Get Network ID	Ca	Ca	Ca	Ca	
0x0F	Query Hop	*Na*	Og	**Ma**	Og	
0x10	Resolve UUID	*Na*	Og	**Ca**[Oa1]	Og	1. Mandatory if device and/or any connected devices connects to multiple buses.
0x11	Query rate limit	Ca[1]	Og	Ca[1]	Og	1. Mandatory if device can support more than 1 request at a time.
0x12	Request TX rate limit	Oa	Og	**Oa**[Ma]	Og	1. Bridges must allow for endpoints to change their rate limits.
0x13	Update rate limit	Oa	Og	Oa	Og	
0x14	Query Supported Interfaces	Oa	Og	Oa	Og	

Key for Endpoint/Bridge columns:

Ma = mandatory to accept
Mg = mandatory to generate
Oa = optional to accept
Og = optional to generate
Ca = conditional to accept (see notes)
Cg = conditional to generate (see notes)
Na = not applicable to accept

Ng = not applicable to generate

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.4 SPDM

The OCP profile for accelerator attestation relies on the upcoming (post 1.0) version of the OCP Attestation of System Components v1.0 to define SPDM requirements for GPUs.

The support for sending 'RawBitStream' as part of Firmware Measurement has been added in the SPDM over Redfish feature under section 4.3.

3.5 PLDM

This section describes how to model a GPU/accelerator PCI-e adapter using PLDM Monitoring and Control protocol. It provides general guidelines on how to model the adapters and their components so that the CSPs can implement a common and generic framework for monitoring, control, and component firmware updates across devices and vendors.

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

The guidelines are based on the PLDM standards as defined by the DSP0248_1.1.0.

3.5.1 Discrete GPU/Accelerator Modules

The block diagram below is a generic representation of an accelerator adapter and can vary based on the vendor. Here, the discrete accelerator device represents the overall adapter and enclosed components as follows:

- MC: This is the management controller which implements PLDM and communicates to the Host BMC
- GPUx: One or more GPU/accelerator controllers
- InterLink: one or more controllers that enable a high-speed connection between GPUs
- PCI-e switch/controller

The block diagram also lists a subset of sensors/effecters that are supported by various components. The following sections describe how to model various sensors and controls via PLDM sensors, effecters and corresponding platform descriptor records (PDRs).

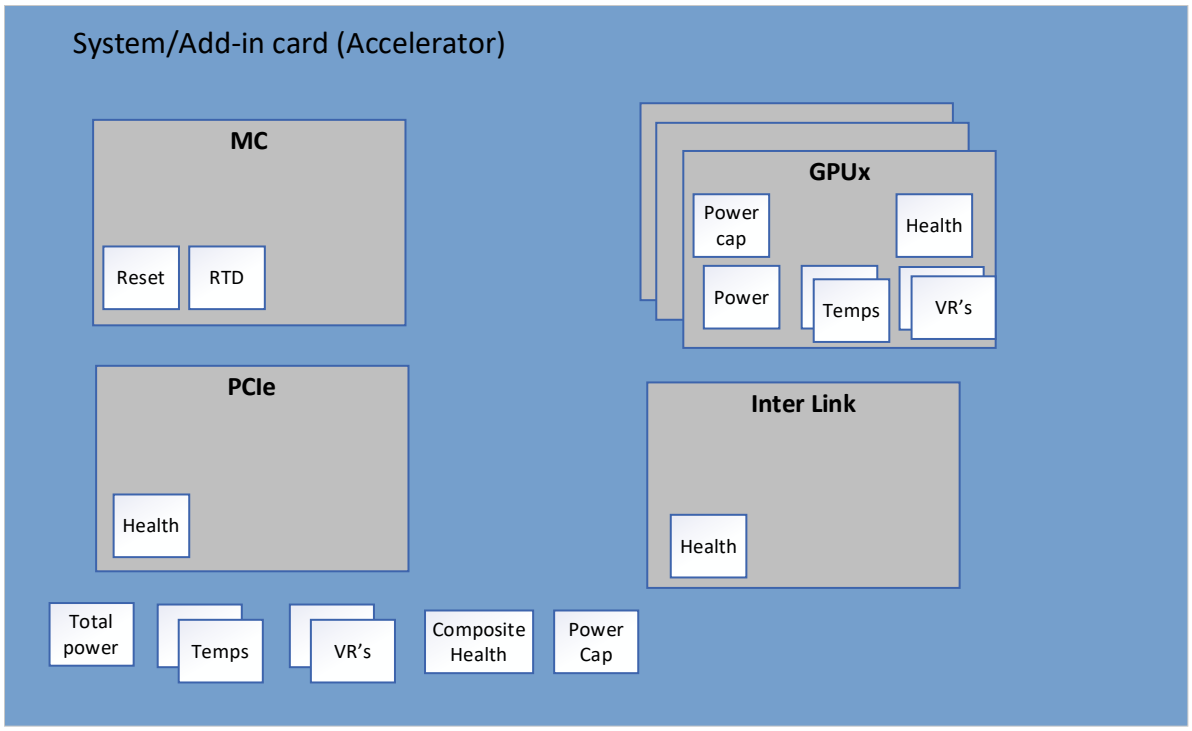


Figure 4 – PCI-e CEM Card

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.5.2 PLDM Sensors & Effecters

The section below provides high-level details of sensors and effecters and presents a minimum set of requirements.

3.5.2.1 Sensors

The implementation must support both numeric and state sensors. At a minimum the sensors listed below are required if present in the hardware design.

The temperature and power sensors are standard numeric sensors supported by the modules whereas the composite health sensor represents the overall health of the adapter.

Table 6 – PLDM Sensors and Types

Sensors	Sensor Type	Add-in Card	MC	GPU	Link	PCI-e
Temp	Numeric	•		•	•	
Power	Numeric	•		•		

Total power	Numeric	•				
VRs	Numeric	•		•		
Composite Healthget	State	•				
Health	State	•	•	•	•	•

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.5.2.2 Effecters

The implementation must support both numeric and state effecters. At a minimum, the effecters listed below are required if supported by the hardware design.

The power cap effecter must be supported at the discrete accelerator device (adapter) level and is optional at the GPU/accelerator module. The reset and reset to defaults (RTD) effecters must be supported by the management controller.

Table 7 – PLDM Effecters

Effecters	Add-in Card	MC	GPU	Link	PCI-e
Power Cap	•		•		
RTD		•			
Reset		•			

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.5.3 Entity Association

The following section describes how to model the entity association PDRs. These PDRs represent modules in a hierarchical model. The sections describe both physical and logical associations.

3.5.3.1 Physical Entity Association

These physical association PDRs are used to describe physical components and their associations. The entity association PDR uses the following references to describe the modules and their association as defined in DSP0248:

- Container ID (an opaque number)
- Contained ID (an opaque number)
- Entity type: The entity type ID are as defined in DSP0248
- Entity instance: Module instance such as “GPU 0” or “GPU 1”

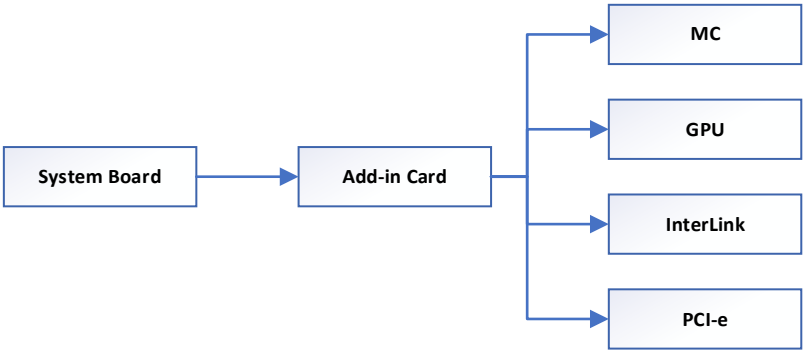


Figure 5 – Component Hierarchy

In the Figure below, the PDRs on the left side and the right side represent the same component hierarchy except via a single PDR vs multiple PDRs; either implementation is valid.

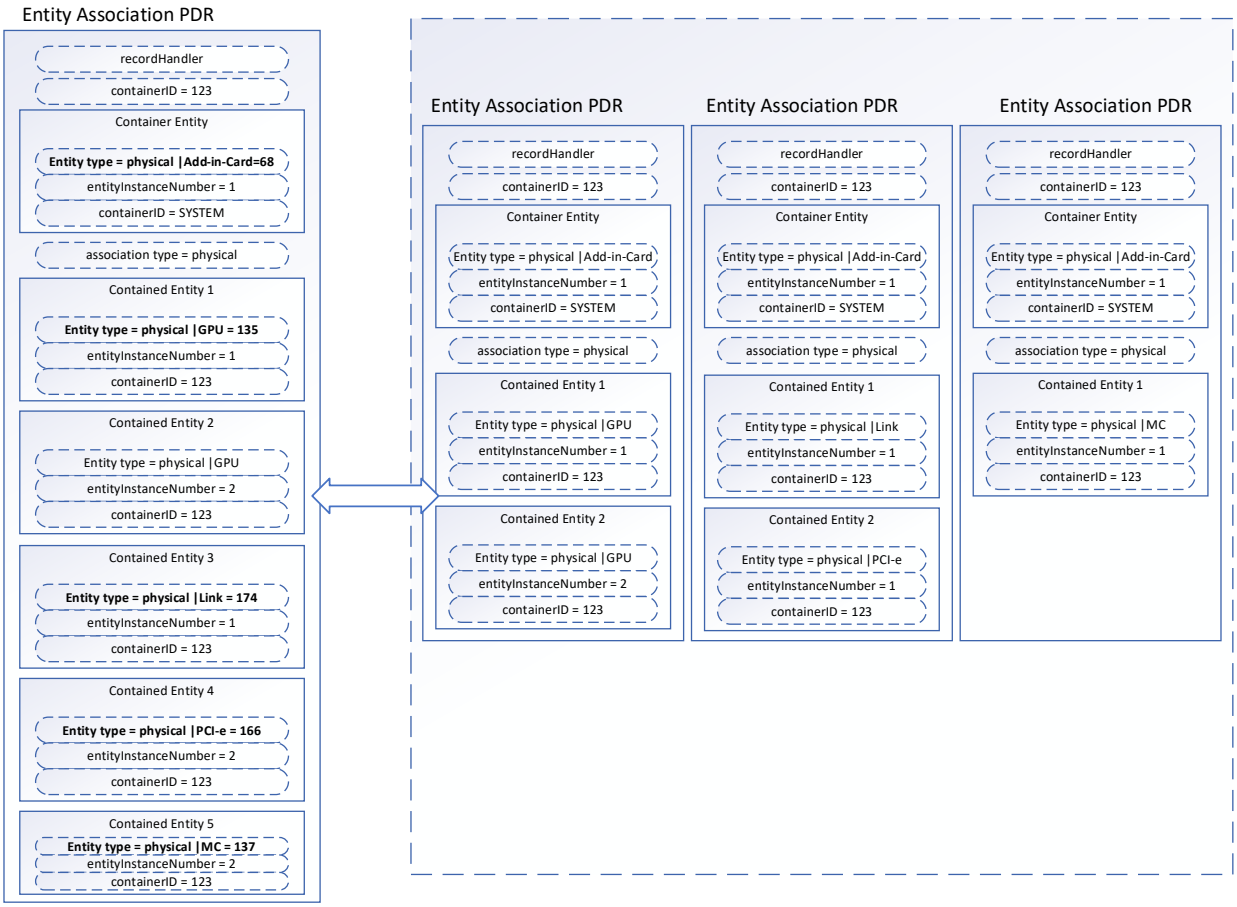


Figure 6 – Entity Association PDRs

3.5.3.2 Logical Entity Association

Logical PDRs are used to associate a set of physical components to a logical entity. In this example the logical entity is “cooling domain”. This helps to group a set of sensors (in this example temperature sensors) of various physical modules to a single logical entity. This will help the management controller to associate these groups of sensors to the thermal algorithms used for fan and/or power control without requiring prior knowledge.



Figure 7 Logical Entity Association

3.5.4 Sensor & Effecter Entity Mapping

The following section provides a list of examples showing how to map various sensors and effecters to entity association PDRs so that the Host BMC can map the sensors to the actual physical modules.

3.5.4.1 Numeric Sensors

In the following example, a temperature sensor with sensor ID 10 belongs to GPU 1 while sensor ID 20 belongs to an InterLink module.

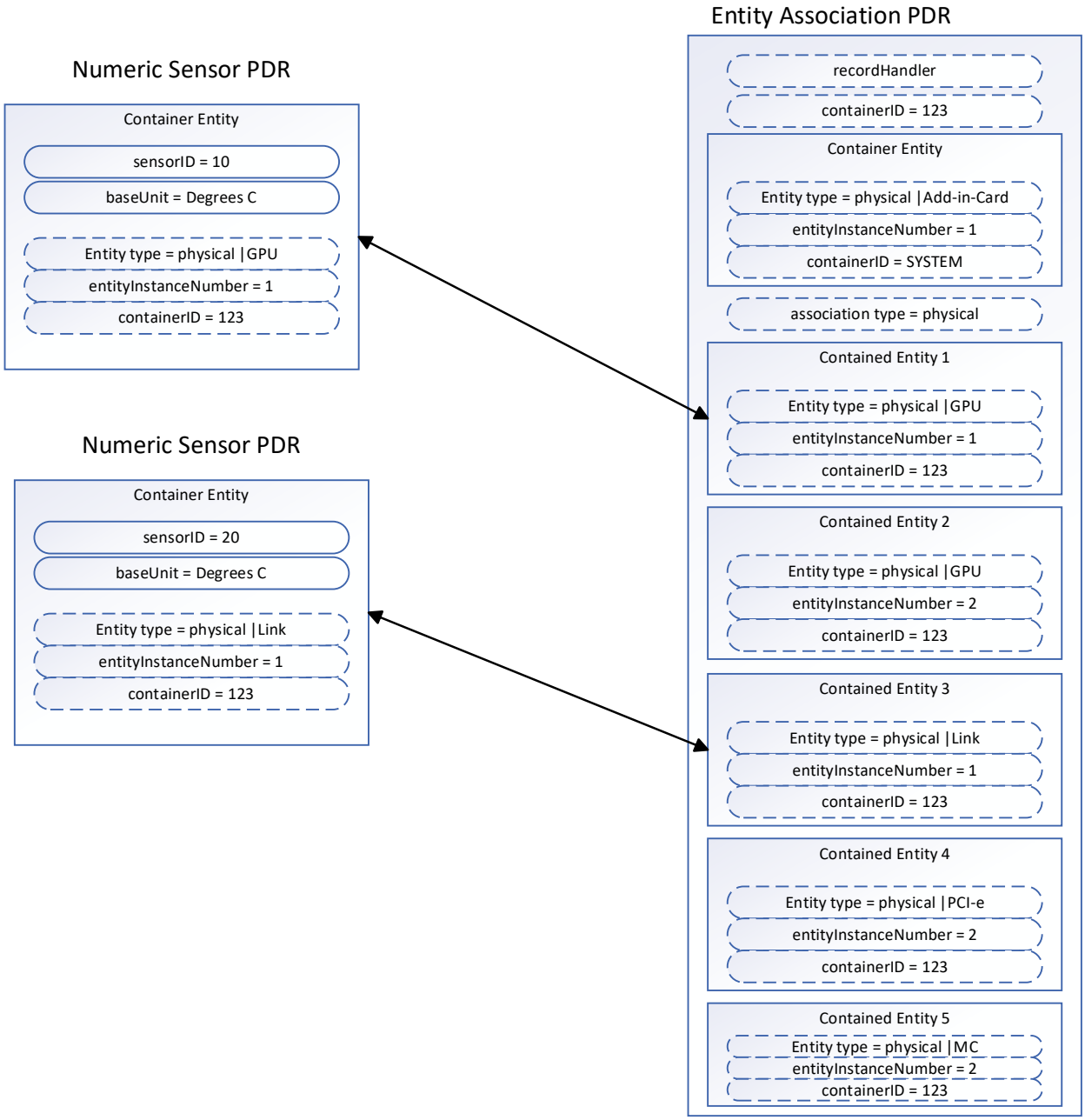


Figure 8 – Numeric Sensor PDRs

3.5.4.2 State Sensors

In this example, the Health state sensor is associated to an Add-in card to represent the overall health.

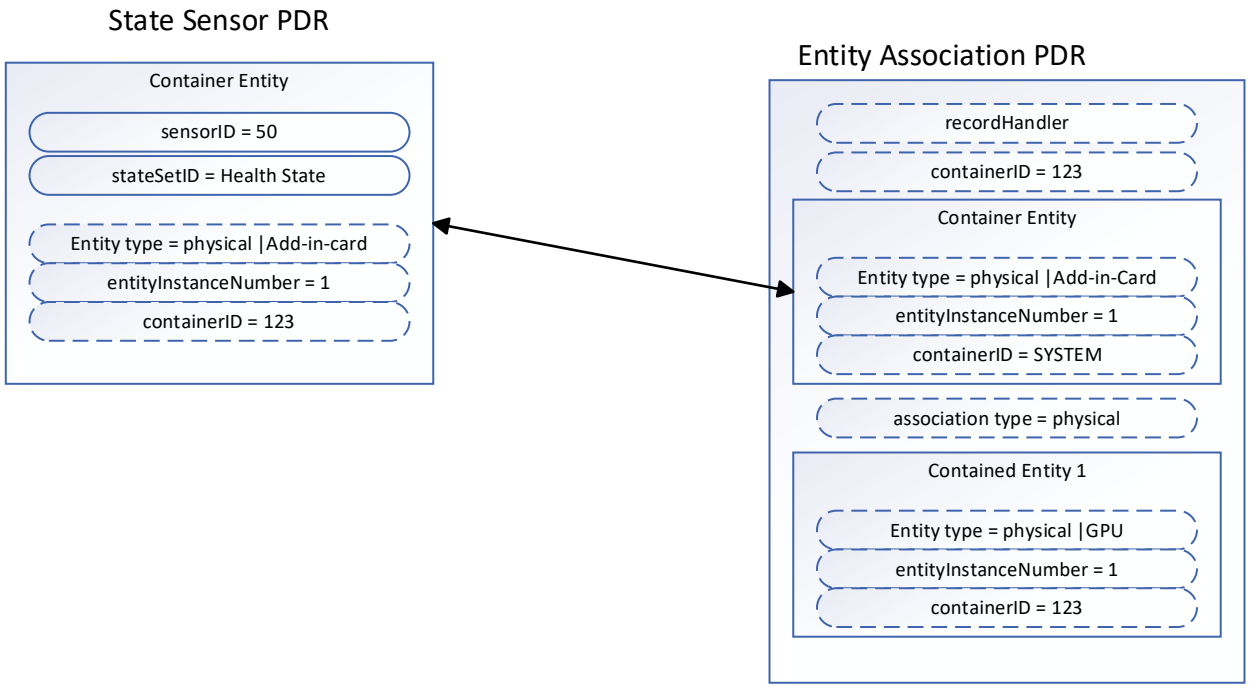
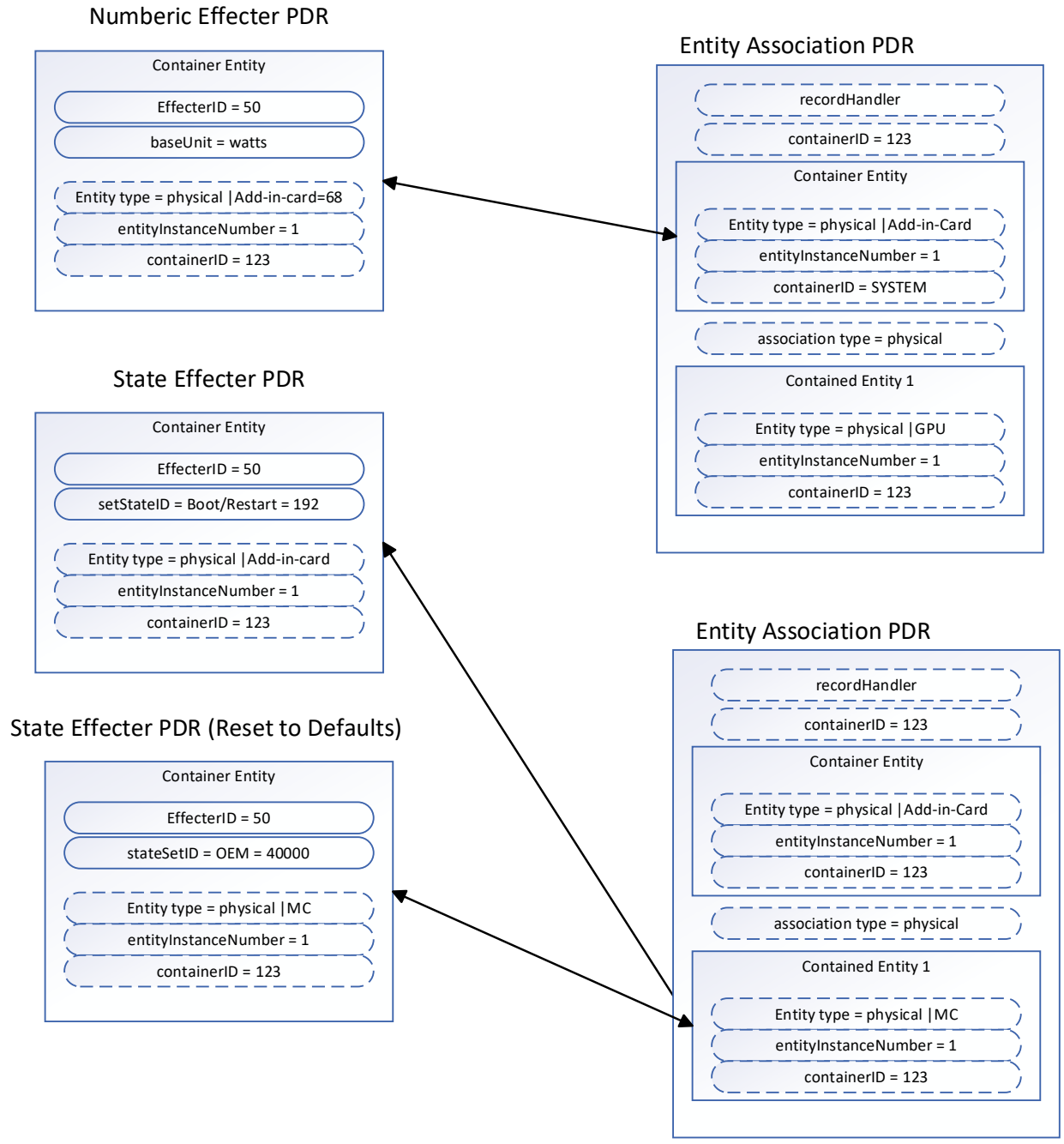


Figure 9 – State Sensor PDR

3.5.4.3 Effecters

In this example, the numeric effecter with ID 50 represents the overall power cap at the Add-in card level.

Likewise, the State Effecters “Reset” and “Reset to defaults” are associated to the management controller.



The Reset to Defaults state effecter shall use OEM State Set 40000, and shall support the enumerated value “1” indicating “Reset to Default”. No other values are required to be supported in this state set.

Figure 10 – Effecter PDR

3.5.4.4 Logical Sensor Mapping

The mapping below associates sensor ID 15 of GPU 1 and sensor ID 25 of the InterLink module to the cooling domain entity.



Figure 11 Logical Sensor Mapping PDRs

3.5.5 PLDM Type IDs

The following table lists the Type IDs used to represent the Accelerator adapter as per DSP0249.

Table 8 – PLDM IDs

Component	Entity	Entity ID
Management Controller	Management controller	137
Add-in card	Add-in card	68
GPU	Accelerator	149
InterLink	Inter-processor bus	174

PCI-e	PCI Express Bus	166

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.6 MCTP OEM VDM

This section lists a reference design for how a device may augment telemetry data over MCTP when existing PLDM models are insufficient. It should be expected that device vendors will have additional OEM telemetry that will be harvested by the management controller that doesn't meet the existing definition of the PLDM Type 2 ecosystem. Fundamentally, the OCP group desires to push PLDM DMTF standards as quickly as possible but acknowledges that an OEM path for passing data is unavoidable.

3.6.1 Management Protocol Structures

This section describes two MCTP OEM VDM bindings, and device vendors may select the binding appropriate for their use case.

There are two forms of OCP VDM bindings defined:

- PCI vendor defined messages, utilizing the device vendor's PCI Vendor ID.
- IANA enterprise ID vendor defined messages, utilizing OCP's IANA enterprise ID (42623).

The subsequent subsections describe the request, response, and event message structures relative to the OCP VDM binding shown below, and the request, response, and event messages are identical other than the choice between PCI Vendor ID and IANA enterprise ID for the MCTP VDM header.

The supported MCTP OEM VDM binding for a particular device may be discovered using the MCTP Get Vendor Defined Message Support command. The device shall return either PCI Vendor ID-based messages support using the vendor's PCI Vendor ID, or shall support IANA Enterprise ID-based messages using OCP's IANA enterprise ID.

A device may also indicate support for both. In the event, that an implementer has vendor defined messages where byte 5 of the "OCP VDM Binding using PCI VendorID" is in the range of 0x80-0xFF and does not conform to the Header Format of byte 5 defined in this specification, the following guideline shall be followed:

- All OCP Header compliant messages, whether OEM or OCP defined, shall use the OCP's IANA Enterprise ID.

A device shall indicate that it supports OCP Header Compliant Messages, but returning 0x8000 in the last 2 bytes of the Get Vendor Defined Message Support (See section 12.9 of DSP0236)

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.6.1.1 Conventions

- Byte order: As per DSP0236, the standardized MCTP header is encoded in Big Endian. This specification requires that non-header byte ordering of multibyte numeric fields or multibyte bit fields is Little Endian.

3.6.1.2 Data Type Enumeration

Table 9 – Data Type Enumeration

#	Type	Explanation
0	bool8	A Boolean value represented using an unsigned 8-bit binary integer where 0x00 means False and any non-zero value means True.
1	enum8	A sequential enumeration, starting from 0 as the default, with optional numeric declarator. The number 8 indicates that the enumeration is encoded using an 8-bit binary number.
2	UINT8	Unsigned 8-bit integer
3	INT8	Signed 8-bit integer
4	UINT16	Unsigned 16-bit integer
5	INT16	Signed 16-bit integer
6	UINT32	Unsigned 32-bit integer
7	INT32	Signed 32-bit integer
8	UINT64	Unsigned 64-bit integer
9	INT64	Signed 64-bit integer
10	bitfield8	8-bit bitfield
11	bitfield32	32-bit bitfield
12	S24.8	Signed fixed point number with 24 bits integer part and 8-bit fractional part
13	String	ASCII encoded sequence of characters with NULL termination
14	Character array	ASCII encoded sequence of characters without NULL termination

15	DateTime	ASCII encoded sequence of characters in ISO 8601 format in the form yyyy-mm-ddThh:mm:ssZ. If the time is unknown, then it may be replaced with 00:00:00Z.

3.6.1.3 OCP VDM binding using PCI Vendor ID

Table 10 – OCP VDM Binding Using PCI ID

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	IC	Message type = 0x7E							PCI vendor ID																RQ	D	Rsvd	Instance ID				
5-8	OCP	OCP Type			OCP Version				Vendor Message Type								Message payload															

3.6.1.3.1 Field Descriptions for OCP VDM binding using PCI Vendor ID

Table 11 – OCP VDM Binding Filed Descriptions

Field Code	Field Size	Description
IC	1 bit	IC. Integrity check.
Message type	7 bits	Message Type = 0x7E.
PCI vendor ID	2 bytes	PCI defined vendor ID of the device vendor
RQ	1 bit	Request bit. This bit is used to differentiate between requests and other kinds of messages. This bit will be set to **1** for requests and events and to **0** for other messages.
D	1 bit	Datagram bit. This bit is used to indicate whether the instance ID is being used for asynchronous notifications (events) or for request/response tracking. This bit will be set to **1** for asynchronous notifications (events) and **0** for requests and response messages. The following table describes valid RQ and D bit combinations. RQ = 0, D = 0 – Response message RQ = 0, D = 1 – Event acknowledgment RQ = 1, D = 0 – Request message RQ = 1, D = 1 – Event message

Rsvd	1 bit	Reserved field. Must be zero.
Instance ID	5 bits	Instance ID. This field is used to uniquely identify a new instance of a request sent to the same MCTP endpoint and to match a particular instance of a request with its response. Retries shall carry the same instance ID as the original request.
OCP	1 bit	OCP designator. For all conformant OCP VDM messages this bit would be set to **1**.
OCP Type	4 bits	OCP management interface type. This field shall be **1** for OCP VDM messages.
OCP Version	3 bits	OCP management interface version. This field shall be **1** for OCP VDM messages.
Vendor Message Type	1 byte	Represents the namespace for a group of related messages.
Message payload	Variable	Message structures. See the subsections below for details.

3.6.1.4 OCP VDM binding using OCP's IANA Enterprise ID

Table 12 – OCP VDM Binding Using IANA

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	IC	Message type = 0x7F							IANA Enterprise ID (MSB) = 0x7F								IANA Enterprise ID = 0xA6								IANA Enterprise ID = 0x00							
5-8	IANA Enterprise ID (LSB) = 0x00								RQ	D	Rsvd	Instance ID					OCP	OCP Type				OCP Version			Vendor Message Type							

3.6.1.4.1 Field Descriptions for OCP VDM binding using OCP's IANA Enterprise ID

Table 13 – VDM binding Using IANA ID Filed Descriptions

Field Code	Field Size	Description
IC	1 bit	IC. Integrity check.
Message type	7 bits	Message Type = 0x7F.

IANA Enterprise ID	4 bytes	IANA Enterprise ID for the Open Compute Project (42623 = 0x0000A67F)
RQ	1 bit	Request bit. This bit is used to differentiate between requests and other kinds of messages. This bit will be set to **1** for requests and events and to **0** for other messages.
D	1 bit	Datagram bit. This bit is used to indicate whether the instance ID is being used for asynchronous notifications (events) or for request/response tracking. This bit will be set to **1** for asynchronous notifications (events) and **0** for requests and response messages. The following table describes valid RQ and D bit combinations. RQ = 0, D = 0 – Response message RQ = 0, D = 1 – Event acknowledgment RQ = 1, D = 0 – Request message RQ = 1, D = 1 – Event message
Rsvd	1 bit	Reserved field. Must be zero.
Instance ID	5 bits	Instance ID. This field is used to uniquely identify a new instance of a request sent to the same MCTP endpoint and to match a particular instance of a request with its response. Retries shall carry the same instance ID as the original request.
OCP	1 bit	OCP designator. For all conformant OCP VDM messages this bit would be set to **1**.
OCP Type	4 bits	OCP management interface type. This field shall be **1** for OCP VDM messages.
OCP Version	3 bits	OCP management interface version. This field shall be **1** for OCP VDM messages.
Vendor Message Type	1 byte	Represents the namespace for a group of related messages.
Message payload	Variable	Message structures. See the subsections below for details.

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.6.1.5 Message Payload Format

The following subsections describe the request, response, and event message structures. The message structure diagrams below are relative to the OCP OEM binding described above which occupies the first 6 bytes of the message.

3.6.1.5.1 Discrete Request Message Format (OCP Version 1)

Table 14 – Request Message Format Version 1

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	…End of OCP message binding																Vendor Command Code								Data Size							
5-N	Data																															

3.6.1.5.1.1 Field Descriptions

Table 15 – Request Message Format Version 1 Filed Descriptions

Field Code	Field Size	Description
Vendor Command Code	1 byte	Vendor command code. Specifies the operation to be performed. Command codes are split into namespaces based on the Vendor Message Type field defined in the OCP MCTP binding.
Data size	1 byte	Size of the trailing command-specific data in bytes. For requests which do not require an additional data payload this field must be 0.
Payload	variable	Optional command-specific data.

3.6.1.5.2 Discrete Request Message Format (OCP Version 2)

Table 16 – Request Message Format Version 2

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	… End of OCP message binding																Vendor Command Code								Reserved							
5-8	Data Size																Data															
9-N	Data																															

3.6.1.5.2.1 Field Descriptions

Table 17 – Discrete Request Message Field Descriptions

Field Code	Field Size	Description
Vendor Command Code	1 byte	Vendor command code. Specifies the operation to be performed. Command codes are split into namespaces based on the Vendor Message Type field defined in the OCP MCTP binding.

Reserved	1 byte	Reserved
Data size	2 byte	Size of the trailing command-specific data in bytes. For requests which do not require an additional data payload this field must be 0.
Payload	variable	Optional command-specific data.

3.6.1.5.3 Discrete Response Message Format

The response message structure follows one of the following formats:

- If a non-aggregate request was completed successfully, the response message will include a completion code of SUCCESS followed by data size and optional data bytes.
- If an aggregate request for multiple telemetry samples was completed successfully, the response message will include a completion code of SUCCESS followed by a telemetry count and a series of telemetry fields.
- If the request was unsuccessful, the completion code will be followed by a Reason code field which may contain extended information about the failure.

The message formats are shown below with their respective variants in the subsections below.

3.6.1.5.4 Discrete Response message with data

Table 18 – Discrete Response Message with Data

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	… End of OCP message binding																Vendor Command Code								Completion Code = SUCCESS							
5-N	Data Size																Data															

3.6.1.5.4.1 Fields Descriptions

Table 19 – Response Message Field Description

Field Code	Field Size	Description
Vendor Command Code	1 byte	OEM Command Code for which this message is the response.

Completion code	1 byte	This field contains the status of the requested operation. See the list of defined Completion Codes table below.
Data size	2 bytes	Data size of the response payload in bytes.
Data	variable	Command-specific response data.

3.6.1.5.5 Discrete Response message with reason code

Table 20 – Discrete Response Message with reason code

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	… End of OCP message binding																Vendor Command Code								Completion Code							
5-8	Reason code																															

3.6.1.5.5.1 Fields Descriptions

Table 21 – Response Message Field Descriptions

Field Code	Field Size	Description
Vendor Command Code	1 byte	Vendor Command Code.
Completion code	1 byte	Status of the requested operation. See the list of defined Completion Codes table below.
Reason code	2 bytes	This field contains the command-specific reason code. This field expands on the status returned in the Completion code field before it.

3.6.1.5.6 Bulk Telemetry Request Message Format

Table 22 – Bulk Telemetry Request Message

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	… End of OCP message binding																Vendor Command Code								Data Size							
5-N	Data																															

3.6.1.5.6.1 Field Descriptions

Table 23 – Bulk Telemetry Request Message Field Description

Field Code	Field Size	Description
Vendor Command Code	1 byte	Vendor command code. Specifies the operation to be performed. Command codes are split into namespaces based on the Vendor Message Type field defined in the OCP MCTP binding.
Data size	1 byte	Size of the trailing command-specific data in bytes. For requests which do not require an additional data payload this field must be 0.
Payload	variable	Optional command-specific data.

3.6.1.5.7 Bulk Telemetry Response Message

Devices may choose to implement the following protocol structure for aggregating multiple telemetry sources into a single response. In this scheme, each data source is assigned a unique Item ID and length which are used to identify the telemetry sample which they precede. A response may consist of multiple such telemetry records. A response may consist of both record formats.

Table 24 – Bulk Telemetry Response Message

<table>
<tr><th rowspan="2">Bytes</th><th colspan="8">Byte 1</th><th colspan="8">Byte 2</th><th colspan="8">Byte 3</th><th colspan="8">Byte 4</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>1-4</td><td colspan="16">End of … OCP message binding</td><td colspan="8">Vendor Command Code</td><td colspan="8">Completion Code</td></tr>
<tr><td>5-8</td><td colspan="16">Telemetry Count = Y</td><td colspan="16">Record</td></tr>
<tr><td>9-12</td><td colspan="32">Record#1</td></tr>
<tr><td>13-16</td><td colspan="24">Record#1</td><td colspan="8">Record#2</td></tr>
<tr><td>17-20</td><td colspan="32">Record#2</td></tr>
<tr><td>21-X</td><td colspan="32">…</td></tr>
</table>

X-X+4	Record#Y	
X+5 .. N	Record#Y	PAD (per MCTP Binding Spec)

3.6.1.5.7.1 Bulk Telemetry Fields Descriptions

Table 25 – Bulk Telemetry Response Message Fields Description (Repeated)

Field Code	Field Size	Description
Vendor Command Code	1 byte	OEM Command Code.
Completion code	1 byte	Used to return the status of the requested operation.
Telemetry count	2 bytes	Count of the number of samples (records) returned by the aggregate request.
Records	N bytes	Records that conform to Compact encoded length format or Byte Length Format

3.6.1.5.7.1.1 Compact Encoded Length Format

Table 26 – Compact Encoded Length Format

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	item ID								B = 0	Reserved			Enc. Len			V	Data															
4-n	Data ...																															

Table 27 – Compact Encoded Length Field Description

Field Code	Field Size	Description
Item ID	1 byte	Unique ID that identifies the telemetry response origin/source, specific to the command.
B	1 bit	Indicates the Length format used for this record. Must be 0

Rsvd	3 bits	Reserved for future use. Must be 0.
Encoded Length Value	3 bits	Encoded Length Value of the data sample in bytes as a power of 2. e.g.: Value 0 = Encoded Length of 2^0, Value 7 = Encoded Length of 2^7.
Valid	1 bit	Valid bit. If the valid bit is unset the subsequent data shall be ignored by the requestor as invalid.
Data	Variable	Telemetry data.

3.6.1.5.7.1.2 Byte Length Format

Table 28 – Byte Length Format

Bytes	**Byte 1**								**Byte 2**								**Byte 3**								**Byte 4**							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1-4	item ID								B = 1	Reserved			Enc. Len=0			V	Length															
4-n	Data …																															

Table 29 – Compact Encoded Length Field Description

Field Code	Field Size	Description
Item ID	1 byte	Unique ID that identifies the telemetry response origin/source, specific to the command.
B	1 bit	Indicates the Length format used for this record. Must be 1
Rsvd	3 bits	Reserved for future use. Must be 0.
Encoded Length Value	3 bits	Always 0
Valid	1 bit	Valid bit. If the valid bit is unset the subsequent data shall be ignored by the requestor as invalid.
Length	2 bytes	Length in bytes of data
Data	Variable	Telemetry data.

3.6.1.5.7.2 Example

An example of the aggregate response to combine temperature samples is shown below. The example message encodes the following telemetry samples into a single response:

Table 30 – Aggregate Response Example

Item ID	Name	Data	Length (Len)	Valid (V)
0	Sensor 0 temperature	T0	4 bytes	Y
1	Sensor 1 temperature	T1	4 bytes	Y
3	Sensor 3 temperature	T3[1]	2 bytes	N
5	Sensor 5 temperature	T5	4 bytes	Y

Table 31 – Aggregate Response Example

Byte 1								**Byte 2**								**Byte 3**								**Byte 4**							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Telemetry count = 4																Item ID= 0								Rsvd				Len = 2			V= 1
T0 (byte 0)								T0 (byte 1)								T0 (byte 2)								T0 (byte 3)							
Item ID = 1								Rsvd				Len = 2			V= 1	T1 (byte 0)								T1 (byte 1)							
T1 (byte 2)								T1 (byte 3)								Item ID= 3								Rsvd				Len = 1			V= 0
T3 (byte 0)								T3* (byte 1)								Item ID= 5								Rsvd				Len = 2			V= 1
T5 (byte 0)								T5 (byte 1)								T5 (byte 2)								T5 (byte 3)							

3.6.2 Completion Codes

Table 32 – Completion Codes

Value	Name	Description
0x0	SUCCESS	The command completed successfully.
0x1	ERROR	Generic error code. This code shall not be returned when a specific error code is available.
0x2	ERR_INVALID_DATA	The request payload contained invalid data or illegal value.
0x3	ERR_INVALID_DATA_LENGTH	The requested data length does not match command specification (too long or too short).
0x4	ERR_NOT_READY	The responder is in a transient state where it is not ready to process the request.
0x5	ERR_UNSUPPORTED_COMMAND_CODE	The command code is not supported for this Vendor message type.

[1] Invalid value.

0x6	ERR_UNSUPPORTED_MSG_TYPE	The Vendor message type used in the request is not supported by the responder.
0x7 – 0x7C	RESERVED	Reserved for future use.
0x7D	ACCEPTED	Response has been accepted. Used for commands that have a delayed response that exceed the base timing.
0x7E	ERR_BUSY	The End Point is busy and cannot accept new requests
0x7F	ERR_BUS_ACCESS	Responder encountered an internal bus error.
0x80 – 0xFF	Command specific	This range of completion code values is reserved for values that are specific to a particular request message.

3.6.3 Reason Codes

Table 33 – Reason Codes

Value	Name	Description
0x0	NONE	No additional reason code is available.

3.6.4 OEM Events

Events are asynchronous notifications that are issued by the managed device. The following message format shall be used by the endpoint initiating the event, if a platform implementation allows for Asynchronous notifications.

Table 34 – OEM Event Message Format

Byte 1								**Byte 2**								**Byte 3**								**Byte 4**							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
OCP Binding																RSVD			ACKR	Version				Event ID							
Event Class								Event State																Data size							
Data (zero or more bytes)																															

3.6.4.1 Fields Descriptions

Table 35 – OEM Event Message Fields Descr

Field Code	Field Size	Description
Rsvd	3 bits	Reserved for future expansion.
ACKR	1 bit	Acknowledgement requested. This field may be used to request an optional ACK for event delivery: 0 – Acknowledgement message is not required 1 – Acknowledgement message is required
Version	4 bits	Event payload version.
Event ID	1 byte	Event identifier. This field is unique within a given Vendor Message Type. Multiple events with the same Event identifier may be distinguished using the Instance ID field in the OCP binding.
Event Class	1 byte	Category of the reported event. This field shall be used to report additional information about the conditions in which the event is generated. See table below for the list of valid event classes.
Event State	2 bytes	Additional information about the event. This field is dependent on the Event Class.
Data Size	1 byte	Size of the event payload (Data) in bytes. This field may be 0 for events without any payload.
Data	Variable	Event-specific payload.

3.6.4.2 OEM Event Classes

The following event classes are defined in this version of the specification.

Table 36 – OEM Event Classes

Event Class	Value	Description
General	0	This category shall be used for general purpose events. Event State will always be **0** for a general event in this version of the specification.
Assertion/Deassertion	1	The category shall be associated with transient conditions that may generate both assertion and de-assertion events. The Event State field may carry more information about the previous and next states as defined by the Event definition.

Vendor defined	128-255	Reserved for vendor use.

3.6.4.3 OEM Event Acknowledgement

By default, events are unacknowledged datagram messages and do not require any action on behalf of the management controller.

However, the device may support a method of generating acknowledged events for e.g., to require delivery guarantees for critical events. In such cases, the device may request an acknowledgement (ACK) from the management controller upon receipt of the event message. This is done using the ACKR flag in the Event message body.

The following requirements apply for acknowledged event messages:

- Restrict to only one event receiver.
- There can only be a single event waiting for an ACK at a time. Unacknowledged events can continue to be generated.
- When an ACK is not received with the acknowledgment timeout duration the device must retry delivery of the event message up to NRETRY times. If a message cannot be delivered after this retry period, the event is considered expired and may be dropped by the device.

The following message format is used for ACK messages which are differentiated based on the RQ and D fields in the OCP binding.

Table 37 – OEM Event Acknowledgement Message Format

Bytes	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1-4	IC	Message type = 0x7E							PCI vendor ID																RQ	D	Rsvd	Instance ID				
5-8	Event ID																															

*PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG

3.6.4.3.1 Field Descriptions

Table 38 – OEM Event Acknowledgement Message Field Descriptions

Field Code	Field Size	Description
RQ	1 bit	Request bit. This bit is used to differentiate between requests and other kind of messages. Would be set to 1b for request messages and event messages. Would be set to 0b for response messages.
D	1 bit	Datagram bit. This bit is used to indicate whether the instance ID is being used for asynchronous notifications (events) or for request/response tracking. This bit shall be set to 1 for event ACK messages.
Rsvd	1 bit	Reserved. Must be 0.
Instance ID	5 bits	Used to uniquely identify a new instance of the API request. This is used to differentiate requests sent to the same MCTP endpoint and to match a particular instance of a request with a corresponding instance of the response.
Event ID	1 byte	Event identifier. This field is unique within a given Vendor Message Type. Multiple events with the same Event identifier may be distinguished using the Instance ID field in the OCP binding.

3.6.5 Timing Specification

Table 39 – Timing Specification

Timing specification	Symbol	Value	Description
Number of request retries	N_{RETRY}	2	The minimum number of times a requester shall retry a request. This number is in addition to the original request. The maximum number of times a request can be retried is unbounded in the current specification.
Request-to-response time	T_{R2R}	100 ms	The maximum duration within which a responder shall begin transmission of a response message. This interval is measured at the responder from the end of reception of a complete request to the beginning of the response's transmission.

Timeout waiting for response	$T_{TIMEOUT}$	300 ms	Typical duration that a requester shall wait for a response message accounting for transmission delay. The interval is measured at the requester from the end of transmission of a request message to the beginning of the receipt of the response.
Retry delay for ERR_NOT_READY	T_{RDY_DELAY}	250 ms	The minimum amount of time the requester shall wait before retrying a request upon receiving the ERR_NOT_READY completion code. This interval is measured from the end of receipt of a response containing the ERR_NOT_READY code and when the requester begins transmitting the next request.
Timeout waiting for long running request	$T_{TIMEOUT_LONG}$	1 second	Typical duration that a requester shall wait for a response message for a long running request.

4 UBB Accelerator Device Interfaces

This section lists the minimum set of protocols and OEM extension/formats/schemas that any GPU/accelerator vendor needs to support on their AMC so that hyperscalers can seamlessly manage these devices without any additional vendor-specific work.

4.1 Redfish

This section describes the high-level overview of Redfish organization. This section also proposes various Redfish resource behaviors and OEM schema extensions. The security aspects of Redfish will be elaborated in future revisions of this specification.

4.1.1 Redfish Tree

The Redfish tree diagram offers a comprehensive overview of the AMC management capabilities and resource organization. It provides an overview and relationships between different components. The resource label “UBB” represents the complete Accelerator subsystem/enclosure within the server and the resource label “OAM_X” represents each of the Accelerator module assembly.

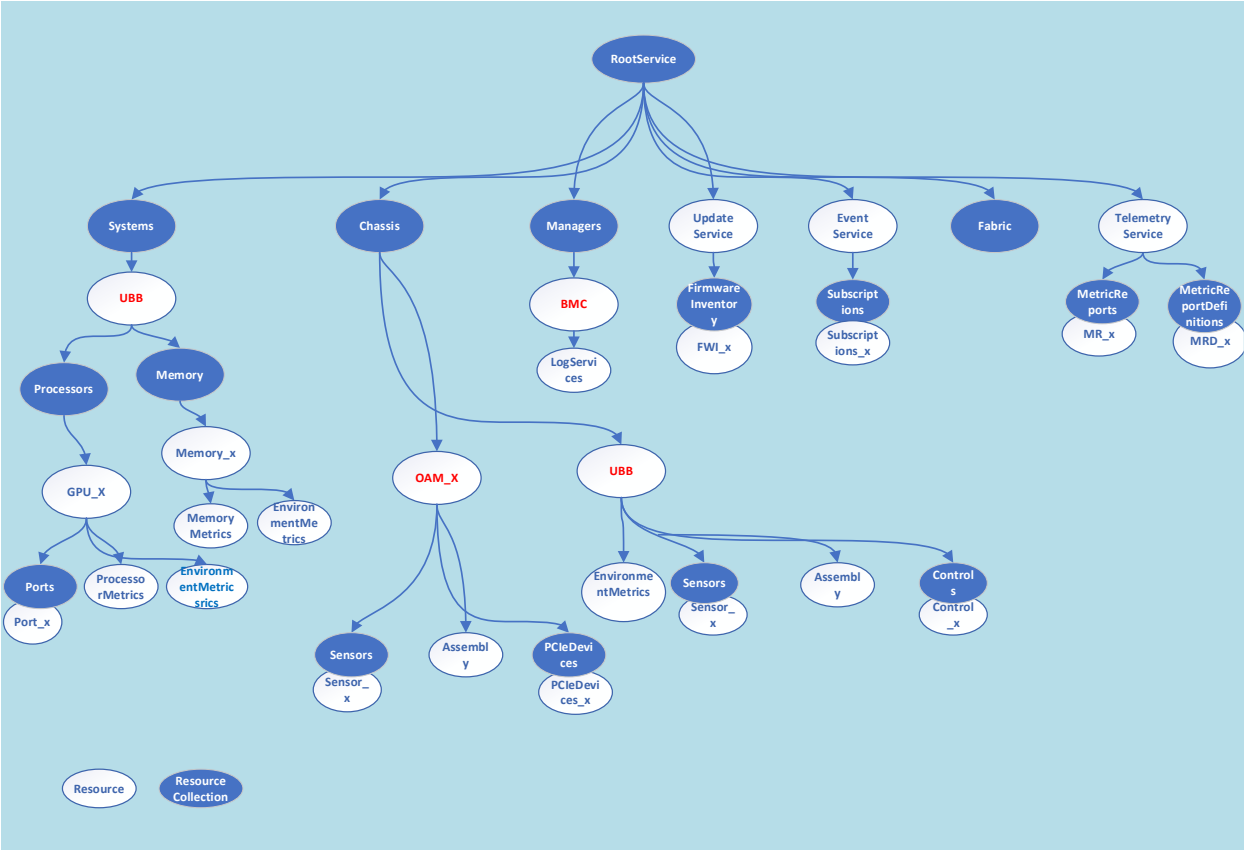


Figure 12 – UBB Redfish Model

4.1.2 Location Objects

The Location object referenced from many Redfish resources is very important in enabling automation at scale. CSPs use various techniques to correlate the contents of the Location object into unambiguous mappings to physical hardware in their data centers.

Table 40 – Location Object Example

```
"Location":
{
  "PartLocationContext": "Backplane 3, Slot 4",
  "PartLocation":
  {
    "LocationType": "Port",
    "ServiceLabel": "Port 1"
  }
}
```

Location property example.

CSPs often rely on a tuple formed by the combination of the `Location.PartLocationContext`, and `Location.PartLocation` to bind entities to external logical management controls. For this reason, the Location resource, and its `PartLocationContext`, `PartLocation.LocationType`, and `PartLocation.ServiceLabel` fields are required by CSPs in any Redfish resource that supports `Location`.

4.1.3 Redfish Interoperability Profile (RIP)

The GPU management Redfish Interoperability Profile (RIP) follows the DMTF Redfish Interoperability Profile standard and provides specifics on the minimum set of Redfish resources and properties required by hyperscalers when managing a UBB-based GPU design.

A supplier shall use the OCP UBB GPU RIP to confirm their accelerator management controller's (AMC) Redfish implementation supports the minimum set of Redfish required to ease integration with hyperscale environments. It should be understood that this RIP is focused on hyperscaler requirements and does not limit the Redfish implementation that a supplier may provide for other classes of customers.

The RIP specification corresponding with this spec release is the *OCP GPU Management RIP 1.0.0. (github link)*

4.1.4 DMTF Redfish Message Registry

The DMTF Redfish Message Registry to support GPU management is incorporated into the Platforms message registry version 1.3.0 and beyond. The purpose of this registry is to ensure a common set of supported messages by GPU suppliers.

Download this registry from the DMTF website in DSP8011 2025.2 or later.

4.2 Out-of-Band Interfaces (MCTP, PLDM, and SPDM)

UBB designs are connected to the enclosing system via high-speed networking, where Redfish is the interface, and out-of-band connections including I2C/I3C, where MCTP, PLDM, and SPDM are the interface protocols.

The implementation of the out-of-band protocols is same as with discrete GPU/Accelerator devices described in the above sections except that it out-of-band communication is limited to specific purposes such as monitoring high-frequency telemetry for thermal and power related features.

4.3 SPDM Over Redfish

For UBB based GPU systems, leveraging the existing support of Redfish over high-speed medium such as USB for SPDM protocol provides better performance when compared to SPDM over MCTP sometimes with speeds as low as 100 KHz or 400 KHz. Additionally, transferring of any larger payload (For example: Certs, bitstream) over MCTP bus sometimes adds significant performance penalties and this problem becomes multifold in UBB-based products which involve multiple firmware components. To mitigate this, the AMC in UBB platform shall support transferring bitstreams (and Certs) via Redfish over vNIC over USB to the orchestrator/requestor.

Redfish *ComponentIntegrity* schema (v1.0.0 or any later version) does not support 'RawBitStream', while SPDM (DSP0274_1.3.0) GET_MEASUREMENT supports 'RawBitStream' and 'Digest'. To add the support for 'RawBitStream':

- Add an OEM 'OCPGPUMgmtWG' property inside 'Measurements' property and use 'RawBitStream' as the value for 'OCPMeasurementType' within 'OCPGPUMgmtWG' OEM property. See the MeasurementIndex 2 below for example.
- Upon request for 'RawBitStream' for any FW component within the PLDM bundle, responder shall send the Base64 encoded raw bit stream (binary image) in string object via 'Measurement'.
- When 'RawBitStream' is not requested, the standard attributes like 'MeasurementType', and 'MeasurementHashAlgorithm' shall be used to send 'Measurement' (digest).

Excerpt of mock-up which captures the RawBitStream response within the Measurements below:
NOTE: The OCPGPUMgmtWG odata.type is just an example and is not part of any standard Redfish schema.


```
"SPDM":
{
…
"MeasurementSet" :
{
  "Measurements" :
  [
    {
      "LastUpdated" : "2024-10-31T20:14:27-07:00",
      "MeasurementIndex" : 1,
      "MeasurementHashAlgorithm" : "SHA256",
      "MeasurementType" : "MutableFirmware",
      "Oem" :
      {
        "OCPGPUMgmtWG" : {}
      },
      "Measurement" : "cf9baf0ae3f225c51b78d063ae21d0d61d1173e50d76b9ee9bfc60bbd80d5ded"
    },
    {
```

```
      "LastUpdated" : "2024-10-31T20:14:27-07:00",
      "MeasurementIndex" : 2,
      "Oem" :
      {
        "OCPGPUMgmtWG" :
        {
          "OCPMeasurementType" : "RawBitStream"
        }
      },
      "Measurement" : "RawBitStream of entire FW image in String"
    },
    {
      "_comment" : "Measurements repeated for FPGA, OAMx, Retimerx, VR"
    }
  ]
},
…
}
```


"Action URIs" :

- "/redfish/v1/ComponentIntegrity/UBB/Actions/OCPGPUMgmtWG.ComponentIntegrity.v1_0_0.SPDMGetRawBitStream"
- "/redfish/v1/ComponentIntegrity/UBB/Actions/ComponentIntegrity.SPDMGetSignedMeasurements"

5 Conclusion

The management profiles for GPUs in this document provide guidance for how GPU devices can be designed and how CSPs can manage them. These profiles reduce the amount of work required for GPU suppliers to bring new products to market by converging CSP requirements around industry standards. Likewise, these profiles reduce the effort and time required for CSPs to bring new GPU designs to market by enabling a higher level of code and test re-use between different GPU parts and vendors.

6 Glossary

AMC – the Accelerator Management Controller presented by a UBB design. This is the Redfish controlled management controller present on an OCP OAI HGX UBB board.

BMC – Baseboard Management Controller. This is a management controller present within system designs that frequently interacts with the management of discrete accelerator devices and UBBs.

CSP – Cloud Service Provider

Hyperscaler – Cloud Service Provider

UBB – Universal baseboard; a type of accelerator delivered as a large board containing multiple GPUs and high-speed fabric interconnect.

7 References

- OCP GPU & Accelerator RAS Requirements v1.0
- OCP Attestation of System Components
- Platform Management FRU Information Storage Definition rev. 1.3 (IPMI)
- DMTF DSP0236 revision 1.3.1 or later (MCTP)
- DMTF DSP0274 revision 1.2.1 or later (SPDM)
- DMTF DSP0240 revision 1.1.0 or later (PLDM Type 0)
- DMTF DSP0248 revision 1.2.2 or later (PLDM Type 2)
- DMTF DSP0267 revision 1.2.0 or later (PLDM Type 5)
- DMTF DSP0266 revision 1.18.0 or later (Redfish)

8 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:

AMD, Google, Meta, 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.

9 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 toil for both the supplier and the hyperscaler consumers.
- A companion to this effort is an 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 hyperscaler 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.

10 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.