OCP GPU & Accelerator RAS Requirements

3 Author (s):

[Rama Bhimanadhuni, Choudary Maddukuri, Bhushan Mehendale, Venkatesh Ramamurthy **Microsoft**]

[David Nieto, Sujoy Sen **Google**]

[Varinder Singh, Pradeep Kumar Shima, Vishal Jain, Linda Wu **NVIDIA**]

Table of Contents

1 Executive Summary....4

2 License....4

3 About Open Compute Foundation....5

4 Introduction....6

5 Goals....6

5.1 High level requirements to meet the goals:....6

5.1.1 Fault-isolation....6

5.1.2 **Hardware Error resiliency** Minimizing the probability of a VM/Node crash due to hardware faults by detecting, signaling, correcting, and often containing errors rather than forcing system resets. .7

5.1.3 **GPU system level crash dumps** To meet hardware error RCA and SLA times, require GPU system level crash dumps and tools to process them....7

5.1.4 **GPU System level reset cause identification** To identify all the GPU subsystem and reset causes, identify the telemetry, and limit the reset impact to subcomponent level to reduce the blast radius Impact....7

5.1.5 **GPU** screening **tools** GPU screening tools which can detect early part defects which can be integrated into manufacturing and production environments....7

5.1.6 Create open-source software that will behave like a compliance check for suppliers and system integrators to ensure GPU compliance....7

5.2 GPU RAS System level view....8

....8

5.3 Scope of the Specification....8

6 Problem Statement....9

7 Error Injection....11

7.1 Importance of Out of Band Error Injection....11

7.2 Error Injection Requirements....12

7.3 Error Coverage....12

7.4 Error Injection Attributes 13
7.5 Redfish based RAS Error Injection for GPU Accelerators 13
7.6 Device Identification using Redfish device tree 17
7.7 Memory Error Injection Details 17
7.8 PCIe Error Injection Details 19
7.9 Error Injection Secure lock and Unlock Querying unlock Status 22
7.9.1 Unlocking Production Grade Hardware & Firmware: 22
7.9.2 Locking Mechanism 23
8 Error reporting Standards 23
8.1 Requirements 24
8.2 Hardware Error Report Standardization Solutions 25
8.3 GPU/Accelerators System level Errors to CPER/Redfish mapping 26
8.3.1 Redfish Examples 29
9 References 30
10 OCP Tenets 30
11 Appendices 30
11.1 Appendix A - Glossary of Terms 30
11.2 Appendix B - Implementation Considerations 33
11.2.1 Memory Error Injection Schema 33
11.2.2 PCI-e Device Error Injection Schema 35

1 Executive Summary

With the explosion of AI/ML workloads, adoption of GPUs and Accelerators is rapidly accelerating in hyperscale Cloud Data Centers. This has created a need for standardizing some critical RAS (Reliability, Availability and Serviceability) capabilities to provide fast adoption and improved TTM for multiple types of GPUs and Accelerators. Some of the critical variances seen in the GPU and accelerator landscape include some gaps in injecting hardware faults and tools, and error reporting. Hyperscalers need the ability to exercise firmware and Software stacks RAS validation flows with hardware fault injections, to verify hardware error handling and resiliency flows, identify faulty FRU, RCA telemetry.

This document will talk about RAS requirements, Redfish based standardization approaches towards hardware error injection and error reporting for different types of GPUs and accelerators. A set of jointly defined Redfish schemas, and how they benefit from the hardware Error fault Management at hyperscale will be covered. The idea is to leverage the Open Compute Project to disseminate these requirements to GPU and accelerator companies along with hyperscalers so that this mutually benefits both and propels our industry forward.

2 License

3 About Open Compute Foundation

The Open Compute Project Foundation is a 501(c)(6) organization which was founded in 2011 by Facebook, Intel, and Rackspace. Our mission is to apply the benefits of open source to hardware and rapidly increase the pace of innovation in, near and around the data center and beyond. The Open Compute Project (OCP) is a collaborative community focused on redesigning hardware technology to efficiently support the growing demands on compute infrastructure. For more information about OCP, please visit us at http://www.opencompute.org.

4 Introduction

This document will focus on the Reliability, Availability and Serviceability (RAS) of GPUs in the context of their usage with hyperscalers. GPU vendors already ship with certain RAS capabilities on the products, but the methods of inducing or injecting faults to validate RAS scenarios and their detection via telemetry vary widely. This results in hyper scalers spending NRE and time on adapting RAS solutions from each supplier to their infrastructure, increasing the time to market. The purpose of this document is to list the goals and requirements that suppliers of GPUs and accelerators must follow in the future to be compliant with the basic, non-IP needs of hyperscalers, such that the adoption of their hardware into hyperscales will be faster. In turn, suppliers will benefit by being able to seamlessly work with multiple hyperscales in the future.

5 Goals

The goals of the RAS solution are:

1. Improve Node Uptime: The goal is to have low Interruption Rate or better MTBF (Mean Time Between Failures). In terms of system availability, the objective is to reduce the probability of crashes and its impact on the service due to hardware failures. This is achieved by implementing various RAS features that allow minimizing the probability of a crash due to hardware faults by detecting, signaling, correcting, and often containing errors and faults, rather than forcing system resets.
2. Serviceability: Reduce MTTR (Mean Time To Recovery) - Ability to identify faulty components to the level of a Field Replaceable Unit (FRU) identification.
3. Root Cause Analysis (RCA): SLA time – Ability to provide Hardware & Firmware error telemetry to RCA issues and meet the SLA requirements as per severity of the issue.
4. Time to Market (TTM): From a hyperscaler perspective, improve the time-to-market for existing and new accelerator hardware coming from suppliers.

5.1 High level requirements to meet the goals:

5.1.1 Fault-isolation

Define a standardized way of Error reporting and error reporting structures to help FRU identification and Error cause Identification telemetry.

- Ability to map hardware errors to the FRU.

- Report the hardware errors using industry standard specifications like Redfish, IPMI SEL, ACPI APEI error record formats defined in UEFI CPER (Common Platform Error Record).
    1. *FRU Isolation* for all types of Hardware Failures.
    2. Needs *structured data* that describes hardware errors.
    3. Supports filtering and categorization based on error severity and types of errors.
    4. Allow entries to contain an *unbounded* amount of data for deep dive.
- **Error Injection Tools:** GPU vendors need to provide standardized based hardware error injection tools which are compatible with cloud infrastructure to verify Error detection, signaling, logging and containment workflows.

5.1.2 Hardware Error resiliency

Minimizing the probability of a VM/Node crash due to hardware faults by detecting, signaling, correcting, and often containing errors rather than forcing system resets.

5.1.3 GPU system level crash dumps

To meet hardware error RCA and SLA times, require GPU system level crash dumps and tools to process them.

5.1.4 GPU System level reset cause identification

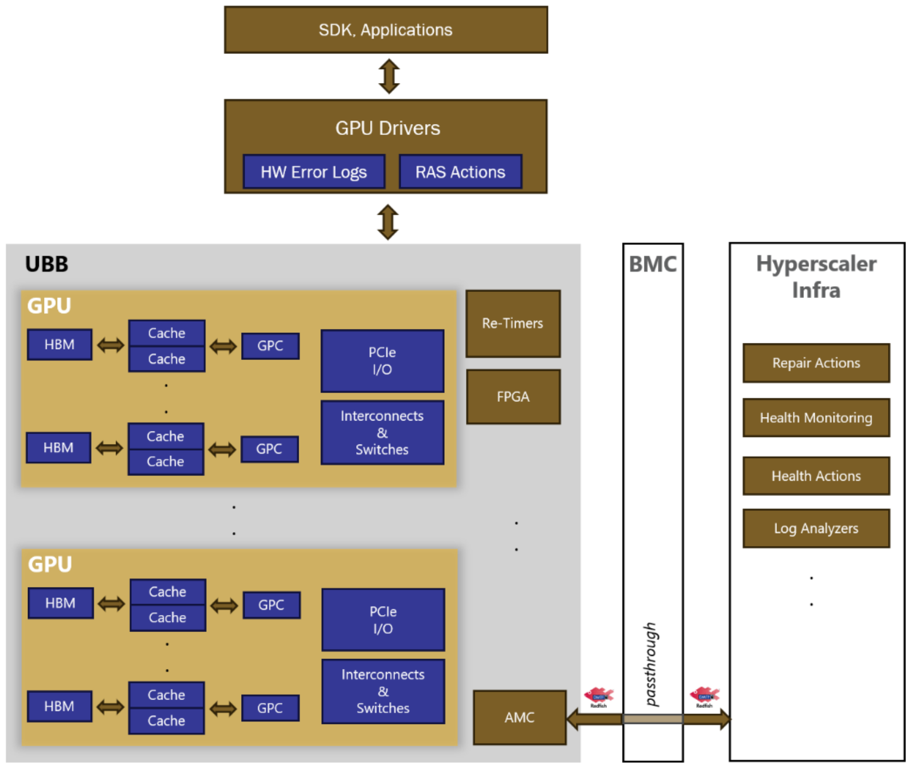
To identify all the GPU subsystem and reset causes, identify the telemetry, and limit the reset impact to subcomponent level to reduce the blast radius Impact.

5.1.5 **GPU** screening **tools**

GPU screening tools which can detect early part defects which can be integrated into manufacturing and production environments.

5.1.6 Create open-source software that will behave like a compliance check for suppliers and system integrators to ensure GPU compliance.

5.2 GPU RAS System level view



5.3 Scope of the Specification

First version of document Scope (Error Injection and Error Reporting)

The scope of this specification is to define error injection capabilities required of a GPU Accelerators. For now, the scope of this is not applicable for Hardware diagnostics or for that matter manufacturing-related activities. The purpose of this is to define a set of capabilities that is required by hyperscalers during the development effort. This implies that suppliers may provide alternate copies of firmware that can be loaded onto GPU accelerators to demonstrate compliance with error injection capabilities.

However, note that error reporting and mitigation aspects of this specification must apply to production level hardware and firmware.

The scope, for now, is limited to out of band error injection.

In the future, this specification may be amended to require error injection capabilities on production quality hardware and firmware from the manufacturer (i.e., components that are running in live, production fleets of hyperscalers in data centers).

6 Problem Statement

Hyperscalers need the ability to adopt newer GPU products from suppliers at a very rapid pace so that they can innovate on their capabilities and provide these to their downstream customers. Hyperscalers face challenges with the integration in accelerators, in part due to the SW complexity and the heterogeneous nature of its implementation.
Hyperscalers need a way to simulate all possible observable faults provided by the accelerator so they can be handled.

Typical pain points encountered by hyperscalers today have been summarized in the following table. These pain points lead to increased Time to Market (TTM) for hyperscalers, and the focus of this effort will be reducing these pain-points:

Issue	**Sub-Issue**	**Customer Workload Impact**	**Hyperscalers View**
GPU not detectable on boot	PCIe Issue Dataplane failure Bootrom Corruption / Bad microcode Interconnect training failures	SLO	These issues are normally detected following firmware updates or during normal operation following. They do not affect runtime workloads but can be challenging on SLO. Preventing these to happen en-masse is critical.
Performance not meeting Criteria	• Thermal issues/Clocking issues • Voltage/noise margin violations • Interconnect flakiness • Memory instability/Correctable memory errors	Yes	• Need monitoring (in-band/out of band). Simulation of thermal/clocking alerts. • We would like deeper data on voltage/noise info. • These tend to happen during runtime execution, and it is difficult to detect the threshold for action.
	• Driver Regression	Partial	• Standard regression benchmarks are normally executed at driver qual.

GPU lost	Uncontained ECC Errors (SRAM/DRAM)	Yes	Usually, indetectable out of band because microcontrollers are affected.
	Row remap ECC Failures	Yes	Improve resilience and reduce sending nodes to repair state
	PCIe Issue (link state, down, correctable, uncorrectable)	Yes	Customers may be executing their workloads when this happens and lose information and data as this situation typically requires a reset of the sub-system or the whole system.
	Re-Timer Issue		
VM Crash	Driver Issues	Yes	Partially detectable by jailing GPU drivers. Open-source is preferable but often not shared.
	GPU lost (as above)	Yes	Need proper driver support of EDC
Data Corruption/ Workload termination	Silent Data Corruptions	Yes	Undetectable, need out-of-band periodic tests.
	ECC Uncontained	Yes	Sometimes puts the GPU ina bad state, but OS not notified.
	Slow Memory Failures (Column failures)	Yes	Pre-flight test passes/ row remapping happens, but it takes a long time to reach RMA threshold poisoning fleet with repeat symptoms.
Power transients	Can cause Performance, GPU loss, etc.	Sometimes	Not enough tests to monitor micro transients exceeding TDP due to Boost control. This leads to insufficient power characterization.
Lack of RCA telemetry	Issue RCA takes time and not able to take right actions	Capacity Issues	In addition to identifying the error, it requires Additional debug telemetry to identify the RCA for the issue, like Error register dumps, Crash dumps.
No Issue Found	Wrong replace of FRUs		Reduce the no Issue found category and avoid or reduce taking wrong actions which leads repeated customer Impact.

7 Error Injection

7.1 Importance of Out of Band Error Injection

Typically, Error handling and recovery code-paths on GPUs are not exercised enough in hyperscaler environments: Hyperscalers cannot rely on the natural fault-occurrence for validation and need something that can be induced on-demand.
GPU systems have complex topologies with multiple components in their path —switches, re-timers, GPUs, CPUs, etc. Thus, Intra-node RAS validation is critical. Secondly, fleet level end-to-end error propagation and containment are critical for hyperscalers to exercise and validate their mitigation/repairability flows.

Today, GPU fault handling is opaque to hyperscalers. When a fault occurs, hyperscalers don't necessarily have the ability to know what exact information to collect to analyze the failure.
Even if hyperscalers can collect any information at all, it may not be possible to disambiguate the data such that they can deduce the remediation information that they need to feed into their repairability workflows. Furthermore, hyperscalers have custom host OS stacks, and by getting the ability to inject errors, it allows them to verify their fault-handling mechanisms.

For hyperscalers today, gleaning this fault information from the GPUs to feed into their fault-handling systems is table stakes—i.e., none of this adds any value to the work hyperscalers do, or offers any incentive for differentiation. By standardizing these methodologies to inject errors and provide the ability to verify them allows the hyperscalers to move fast and improve their TTM. Similarly, it allows suppliers to benefit from consistent interfaces against which they can implement their technology (e.g., re-timer devices, switches, etc.)

Hyperscalers often deal with GPUs falling off the PCIe bus. And there could potentially be several contributors to this issue ranging from the GPU core to PCIe devices in the path. It's important for hyperscalers to identify the contribution of GPUs to this issue. Error-injection allows hyperscalers to verify their designs and characterize and identify such failures along with the area of the failure. Many times, GPU framing errors masquerade themselves as PCIe correctable errors in intermediate switches. Error injections allow the hyperscalers to test end-end error propagation behavior and understand system/workload impact.

Catastrophic errors encountered by GPUs are sent to the host SW stack but are only visible to the guest. Consequently, this information becomes unusable for hyperscalers. Such markers are critical for Hyperscalers to consumer via an out of band methodology to enable them to intervene and remediate the issue without end customer dependency.

7.2 Error Injection Requirements

1. Error injection methods shall provide the following for verification:
    a. Error signaling – Error injection should be able to help with verifying error signaling flows, instead of just the error simulation without signaling flows invoked.
    b. Faulty FRU identification
    c. Verification of the Error Mitigation
    d. Verification of the error flow handling in the platform hardware, firmware, and OS/ drivers.
2. Error injection capability to support different HW error capabilities at the IP Level, SOC Level, and on the platform level (e.g., HBM, PCIe).
3. Error Injection capability to support Error severity for, Uncorrectable, Correctable and Fatal errors.
4. Error injection methods and interfaces shall be abstracted from silicon specific implementations.
5. Error injection support should provide Secure unlock and lock mechanisms.
6. Errors listed in this specification shall be injected solely using the Redfish schema defined in this specification. Additionally, this process should be OS-agnostic.

7.3 Error Coverage

Error type	**Details**	**Priority/Comments**
Memory	GPU SRAM and GPU DRAM (HBM)	Priority DDR memory is part of Host Motherboard excluded from here.
PCIe errors	• PCIe Switches, PCIe Network devices, PCIe End point Devices • PCIe Re-timers • PCie Link	Priority PCIe Re-timers require special attention as Errors are different from the general PCIe Endpoint devices and PCIe Switches PCIe Links will have Link Width, Link Speed, Link down Error considerations.
GPU Core Errors	Internal GPU micro controller exceptions and firmware faults	High Prioirty
GPU Links and Switches	GPU Interconnects, Interconnect Switches	
Platform Specific Errors	FPGA Specific Errors, Thermal, USB VNIC, I2C, I3C	Errors applicable to UBB

7.4 Error Injection Attributes

PCIe Errors Attributes

Attributes	Details	Comments
Device Identification	Uses Redfish based URL	Specifies the Device where to do Error injection
Error Severity	PCIe Correctable PCIe Non-Fatal and PCIe Fatal	Multiple Error severity should be supported
Error Type	**Correctable** e.g., Bad TLP, Bad DLLP, Receiver Error, Reply Timeout Ref [1] **Non Fatal** e.g., Poisoned TLP received, Completion Timeout, Unexpected Completion Ref [1] **Fatal** e.g., Malformed TLP, Flow control Protocol Error, Training Error, Receiver Overflow. Ref [1]	Multiple Error types for each Error severity need to be supported

Memory Errors

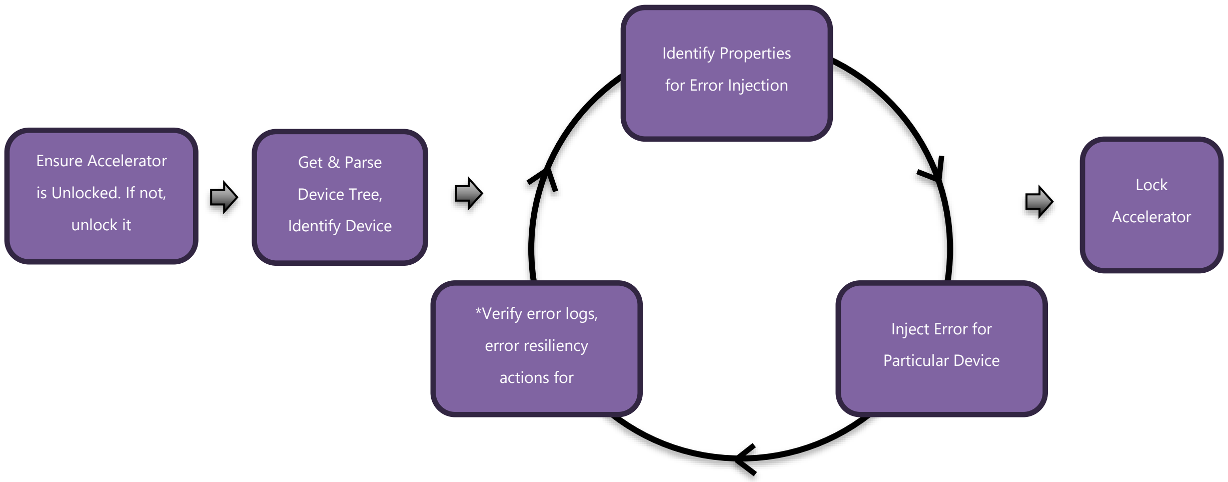
Attributes	Details	Comments
Physical Device Identification	Uses Redfish based URL	Specifies the Device where to do Error injection
Sub Device Identification	Rank, Column, Row Level	Optional
Error Severity	Correctable Uncorrectable	Multiple Error severity should be supported
Address	Memory Address location where to Inject Error	Optional

Memory Error Poison TBD (Planned for next revision of the Spec)

7.5 Redfish based RAS Error Injection for GPU Accelerators

The below section provides details about the error injection at various hardware component levels. The initial document focus provides specific resource examples for Memory and PCIe and the same can be expanded to other resources that the GPU/Accelerator vendors can support as OEM extensions. The errors are injected via Redfish actions.

The steps below provide a high-level flow of Error Injection


Ensure Accelerator is Unlocked. If not, unlock it
Get & Parse Device Tree, Identify Device
Identify Properties for Error Injection
Inject Error for Particular Device
*Verify error logs, error resiliency actions for
Lock Accelerator

1. Device Identification – Using redfish Device Tree
2. Error Injection properties Identification
3. Secure unlock
4. Error Injection for a particular Device
5. Secure Lock
6. Compliance - Verify the Error logs and Error resiliency Actions.

Example steps on how to utilize the error injection using Redfish APIs

1. Do a “GET” on the resource will provide the error injection actions.
   a. Memory error injection actions are available in the Memory resource. The URI format shall follow the patterns defined in the Memory schema.
   b. PCIe error injection actions are available in the PCIeDevice resource. The URI format shall follow the patterns defined in the PCIeDevice schema.

   Clients shall traverse to the Processor resource, then do a GET on the Memory or the PCIeDevice resources in the Links object will find the error injection actions details. Example flows:

   1. Do a GET on a Processor resource:

```
{
 "@odata.id": "/redfish/v1/Systems/{ComputerSystemId}/Processors/{ProcessorId}",
 "Links": {
  "Memory": [ {
    "@odata.id": "<Memory ResourceUri>"
  } ],
  "PCIeDevice": {
   "@odata.id": "<PCIeDevice ResourceUri>"
  },
  ...
}
```

2. Do a GET on a Memory resource:

```
{
 "@odata.id": "<Memory ResourceUri> ",
 "Actions": {
  "Oem": {
   "OCP":{
    "#OcpMemory.InjectCorrectableErrors": {
     "target": "<Memory Resource Uri>/Actions/Oem/OcpMemory.InjectCorrectableErrors",
     "@Redfish.ActionInfo": " <Memory
ResourceUri>/Oem/OCP/InjectCorrectableErrorActionInfo"
    },
    "#OcpMemory.InjectUncorrectableErrors": {
     "target": <Memory ResourceUri>/Actions/Oem/OcpMemory.InjectUncorrectableErrors",
     "@Redfish.ActionInfo": " <Memory
ResourceUri>/Oem/OCP/InjectUncorrectableErrorActionInfo"
    }
   }
  }
 }
 ...
}
```

2. Do a “POST” action will inject the necessary error:
    a. POST {ResourceUri}/Oem/Actions/{ResourceType}.{ActionName}

       Memory error injection examples:

        i. To inject memory poison error (defined in the DMTF standard schema):

```
HTTP POST {ResourceUri}/Actions/Memory.InjectPersistantPoison
{
 “PhysicalAddress”: “0x1000”
}
```

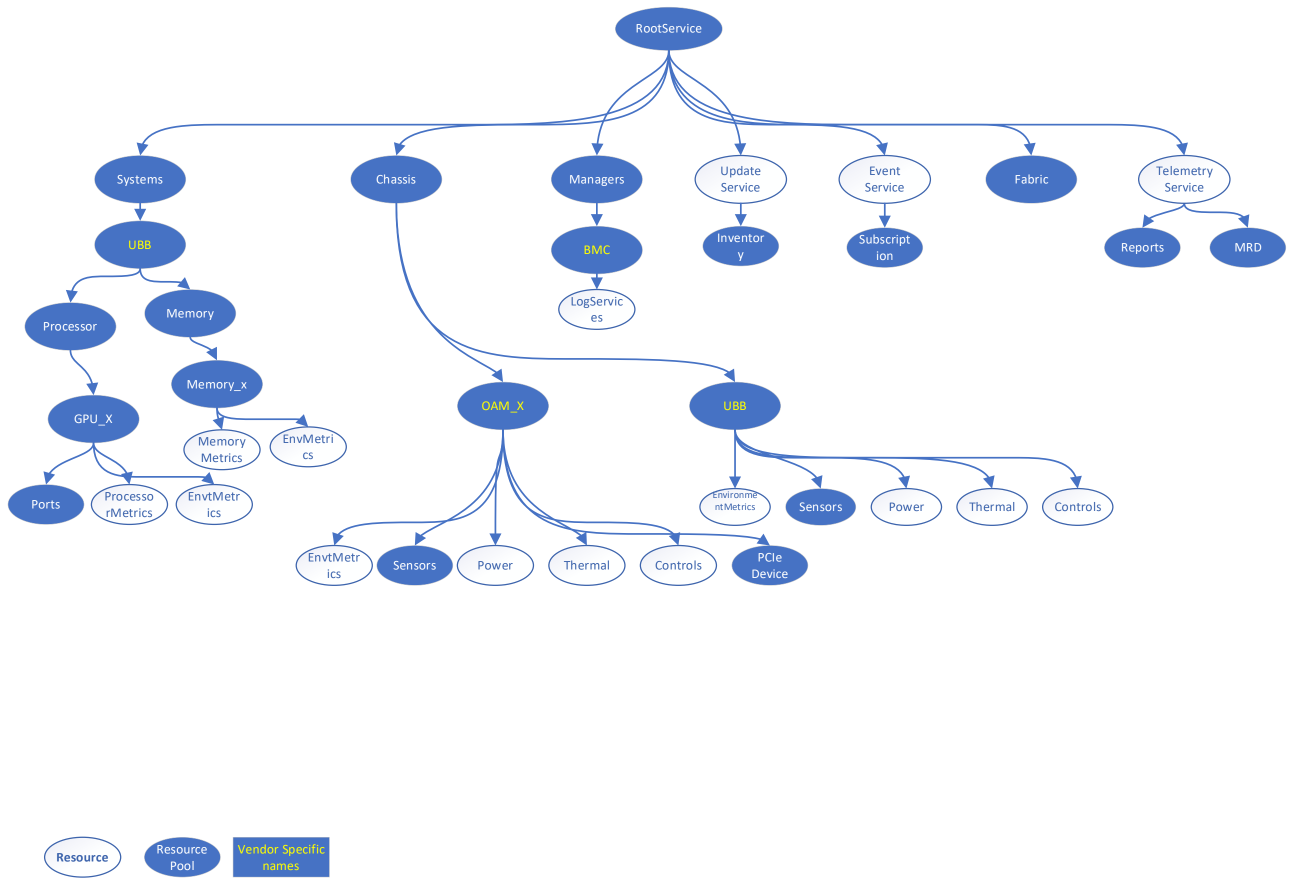
        ii. To inject memory uncorrectable error (defined in the OCP extension):

```
HTTP POST {ResourceUri}/Actions/Oem/OcpMemory.InjectUncorrectableError
{
 “PhysicalAddress”: “0x1000”
}
```

3. Do a “GET” on the metric resource (for example, MemoryMetrics) will reflect the new value in the error counters. For example, the value of “UncorrectableErrorCount” property shall be increased after a successful uncorrectable error injection.
4. In addition, in some cases you will see a corresponding error log in the LogService resource.

7.6 Device Identification using Redfish device tree

Example of a redfish device tree for UBB



7.7 Memory Error Injection Details

The latest schema (V_1_17_0) added a new property "InjectPersistentPoison (Action)" in support of memory error injection. In the same lines the document proposes 2 OEM OCP actions, 1)InjectCorrectableError and 2)InjectUncorrectableError. Below section provides the details

Property Name	Schema(s)	Parameters	Type	Description
InjectPersistentPoison	Memory(Actions)	PhysicalAddress,	Object	Injects poison to the memory address in the memory device.
InjectCorrectableError	OcpMemory(Actions)	PhysicalAddress	Object	Injects correctable errors to the memory address in the memory device.
InjectUncorrectableError	OcpMemory(Actions)	PhysicalAddress	Object	Injects an uncorrectable error to the memory

				address in the memory device.

Note: In future version (0.7), we plan to add additional memory error injection attributes related to device physical location e..g., row, column, bank and rank.

Property Details

InjectPersistentPoison action defined in Memory schema

```
  "InjectPersistentPoison": {
        "description": "Injects poison to a persistent memory address in the memory device.",
         "parameters": {
           "PhysicalAddress": {
              "description": "The device physical address as a hex-encoded string.",
              "requiredParameter": true,
              "type": "string"
           }
        },
        "type": "object",
     },
```

InjectCorrectableError & InjectUncorrectableError actions defined in OcpMemory schema

```
         "InjectCorrectableError": {
                "description": "Injects a correctable error to a specific persistent memory address in the memory
device. ",
               "parameters": {
                      "PhysicalAddress": {
                              "description": "The device physical address as a hex-encoded string.",
                              "requiredParameter": true,
                              "type": "string"
                       }
                },
             },
        "InjectUncorrectableError": {
                "description": "Injects an uncorrectable error to a specific persistent memory address in the
memory device. ",
               "parameters": {
                      "PhysicalAddress": {
                              "description": "The device physical address as a hex-encoded string.",
                              "requiredParameter": true,
                              "type": "string"
                       }
                },
             ]
         }
```

Action Details

```
{
  "@odata.id": "<Memory ResourceUri>”,
  "Actions": {
   "Oem": {
    “OCP”:{
     "#OcpMemory.InjectCorrectableErrors": {
      "target": "<Memory ResourceUri>/Actions/Oem/OcpMemory.InjectCorrectableErrors",
      "@Redfish.ActionInfo": "<Memory ResourceUri>/Oem/OCP/InjectCorrectableErrorActionInfo"
     },
     “#OcpMemory.InjectUncorrectableErrors": {
      "target": "<Memory ResourceUri>/Actions/Oem/OcpMemory.InjectUncorrectableErrors",
      "@Redfish.ActionInfo": "<Memory ResourceUri>/Oem/OCP/InjectUncorrectableErrorActionInfo"
     }
    }
   }
  }
  ...
}
```

7.8 PCIe Error Injection Details

PCIeDevice Error Injection actions defined in OcpPCIeDevice schema

The following section details the error injection to PCI-e devices.

Property Name	Schema(s)	Type	Description
InjectCorrectable	OcpPCIeDevice(Actions)	Object	Injects correctable PCIe errors
InjectUncorrectableNonFatal	OcpPCIeDevice (Actions)	Object	Injects uncorrectable PCIe non-fatal errors
InjectUncorrectableFatal	OcpPCIeDevice (Actions)	Object	Injects uncorrectable PCIe fatal errors

Property Details

```
        …
“Oem”: {
    "OCP": { [
        " InjectCorrectableError ": {
            "additionalProperties": false,
            "description": "Injects Correctable Error to a specific PCIe device.",
             "parameters": {
                 “ErrorType”: {
                      “enum”: [
                        “ReceiverError”,
                        “BadTLP”,
                        “BadDLLP”,
                        “ReplayTimerTimeout”,
                        “ReplayNumRollover”
                       ],
                   “type”:”string”
                   }
               }
           }
        " InjectUncorrectableNonFatalError ": {
            "additionalProperties": false,
            "description": "Injects Uncorrectable NonFatalError to a specific PCIe device.",
             "parameters": {
                 “ErrorType”: {
                      “enum”: [
                        “PoisonedTLPReceived”,
                        “ECRCCheckFailed”,
                        “UnsupportedRequest”,
                        “CompletionTimeout”,
                        “CompleterAbort”,
                        “UnexpectedCompletion”
                       ],
                   “type”:”string”
                   }
               }
            }
        " InjectUncorrectablFatalError  ": {
            "additionalProperties": false,
            "description": "Injects UncorrectableFatalError to a specific PCIe device.",
             "parameters": {
                 “ErrorType”: {
                      “enum”: [
                        “TrainingError”,
                        “DLLProtocolError”,
                        “ReceiverOverflow”,
                        “FlowControlProtocolError”,
                        “MalformedTLP”
                       ],
                   “type”:”string”
                   }
               }
            }
          ]
      }
}
```

Action Details

```
{
 "@odata.id":"<PCIeDevice ResourceUri>",
 "Actions": {
  "Oem":{
   "OCP":{
    "#OcpPCIeDevice.InjectCorrectableError": {
     "target": "<PCIeDevice ResourceUri>/Actions/Oem/OcpPCIeDevice.InjectCorrectableError",
     "@Redfish.ActionInfo": "<PCIeDevice ResourceUri>/Oem/OCP/InjectCorrectableErrorActionInfo"
    },
    "#OcpPCIeDevice.InjectUncorrectableNonFatalError": {
     "target": "<PCIeDevice ResourceUri>/Actions/Oem/OcpPCIeDevice.InjectUncorrectableNonFatalError",
     "@Redfish.ActionInfo": "<PCIeDevice ResourceUri>/Oem/OCP/InjectUncorrectableNonFatalErrorActionInfo"
    },
    "#OcpPCIeDevice.InjectUncorrectableFatalError": {
     "target": "<PCIeDevice ResourceUri>/Actions/Oem/OcpPCIeDevice.InjectUncorrectableFatalError",
     "@Redfish.ActionInfo": "<PCIeDevice ResourceUri>/Oem/OCP/InjectUncorrectableFatalErrorActionInfo"
    }
   }
  }
 }
 ...
}
```

7.9 Error Injection Secure lock and Unlock Querying unlock Status

As you can see in Figure 1 In the above diagram, before errors can be injected, the GPU accelerator (discrete or otherwise) must have the ability to be "unlocked." The supplier may have locked their components from a production perspective with security in mind, and this process will unlock the component and ensure it's ready for injecting errors. More on locking and unlocking (step #4 in the above diagram) is discussed further down in this document.

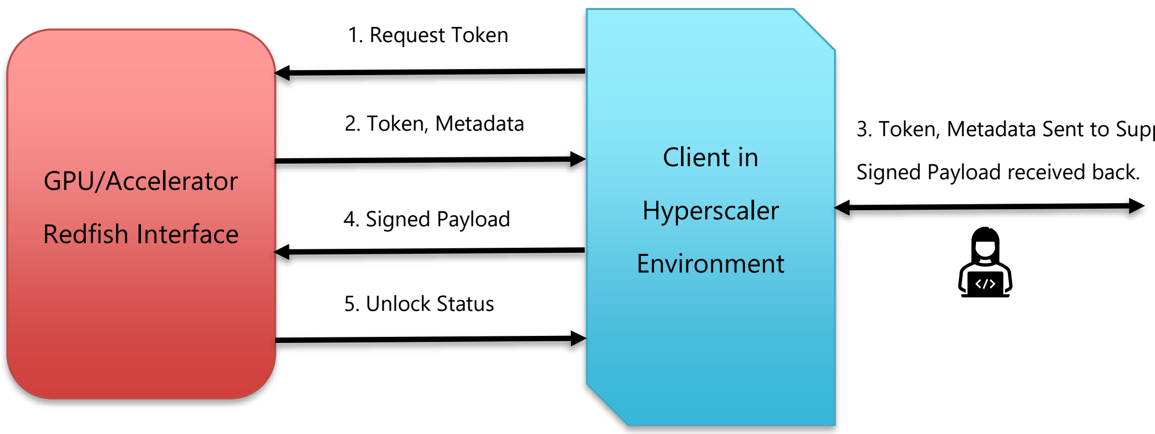
Once the GPU accelerator is ready to receive error injection via Redfish, one can request areas of supported error injection from the GPU accelerator (#2). Using this information, one can in a loop continue to inject errors, and receive telemetry confirmation and mitigation of the injected errors (step #3).

Finally, one should be able to "lock" the GPU accelerator and put it back into production mode.

Suppliers must provide hyperscalers with "development" grade or "debug" grade builds/cards of GPU accelerators. Such hardware and firmware running on it shall be unlocked by default and support the error injection requirements specified in this documentation.

On "production" grade hardware and firmware running on it which is locked by default, suppliers must provide a way to unlock the cards to support error injection. The process is outlined below.

7.9.1 Unlocking Production Grade Hardware & Firmware:


GPU/Accelerator Redfish Interface
1. Request Token
2. Token, Metadata
4. Signed Payload
5. Unlock Status
Client in Hyperscaler Environment
3. Token, Metadata Sent to Supplier,
Signed Payload received back.

As a prerequisite to the unlocking mechanism, the supplier shall ensure that there's a mechanism to query the GPU accelerator for its lock status.

Hyperscaler, through Redfish, requests a token from the accelerator.

1. Hyperscaler requests an unlock token from the accelerator.
2. Hyperscaler receives a device-specific token + metadata (payload) from the accelerator.
3. Through an out of band process, the hyperscalers have the supplier sign the token and get a payload to send to the accelerator to unlock it.
4. Hyperscaler receives this payload and sends it to the accelerator. What this payload is, is left up to the supplier. It may be a small unlock token, or an entire firmware drop. But the supplier must ensure that this payload is only applicable to the hardware it is requested for, and no other GPU accelerator hardware.
5. The accelerator unlocks and sends status.

7.9.2 Locking Mechanism

The only way to re-lock unlocked production-grade hardware is to re-flash it with production-grade firmware. This should be done via a standard firmware update process as specified in the FW Update Specification Note: <Ref to Firmware update specification will be done after OCP spec published.>.

Note that the above unlocking mechanism is expected to be used as a one-off on a case-by-case basis. i.e. it is not intended to be used in production or in the factory during manufacturing. As such the supplier will ensure they take reasonable steps to have their unlocking service available 99% (two nines) of the time.
Note: In the next revision (0.7 spec plan) to add specific lock and unlock flows with schema.

8 Error reporting Standards

Hardware Error reporting formats currently used on the servers deployed in large clusters varies across various CSPs do not have standardized error record formats. The error record format needs to satisfy goals of identifying the FRU which caused the error and provide extensive information about the error which can help identify the first level cause of the error. In some cases, full error root cause may require hardware and software state.
Also, Hardware errors can be reported by different agents like UEFI Firmware, System management controller (e.g., BMC) and OS based on the Error handling Implementation and type of hardware errors. Having the same error formats across different error reporting agents will be helpful in terms of Error log Harvesting tools and Debug.

Currently Hardware error reporting uses two existing Standards based on errors to be reported as described below.

In the first one, IPMI specification-based System Event Logs (SEL) to report hardware errors. But IPMI based SEL have these limitations.

- SEL error records provide primitive data for reporting Hardware Errors.
- IPMI spec does not cover all types of Hardware errors e.g., lack of support for new technology-based errors - CPU Interconnect errors (CXL, UPI, GSMI)
- Also, this IPMI specification is not maintained by the industry anymore, so this specification is becoming absolute.

In place of IPMI specification, DMTF organization has developed a secure and scalable interface specification called *Redfish* for the modern datacenter environments. This specification does not directly define hardware Error formats but provides an infrastructure to define the error records. There is need to define Hardware error standardization.

The second Industry standard is UEFI specification based Common Platform Error Record (CPER) for hardware error reporting. Currently UEFI firmware implementations use these CPER records for sharing the hardware error information to the OS. And the OS also uses CPER records to report hardware errors handled by the operating system.

Hardware error records access through BMC for Out of Band purposes is very critical for fault diagnostics in both non bare metal and bare metal use cases. This is one of the widely used mechanisms currently to harvest error data reliably on scale of systems. Also, BMC based RAS error handling cases are increasing which requires hardware error record format standardization and APIs to read these records.

The standard for error reporting must satisfy the needs that would enable the faster deployment of heterogenous hardware and the minimum cost to implement and maintain.

8.1 Requirements

- The Error record format needs to provide complete and accurate information necessary to fully categorize the error and provide the first level response to the field. This includes identifying the FRU (Field Replaceable Unit) which caused the error.

- The error format must also be structured which simplifies the programming of tools to filter and categorize the errors.
- Hardware errors can be reported by different agents like UEFI Firmware, Management controllers (BMC), GPU Drivers/Software and OS based on the Error handling Implementation and type of hardware errors. Having the same error formats across different error reporting agents will streamline the tools and Easy Error harvesting.
- In some cases, full error root cause analysis may require hardware and software state, and this will create architecture specific unbounded blobs of information that will be required with error messages.
- Also, the extension of categories will keep the standard alive, by allowing new categories for new technologies or methodologies to be added.
- Finally, it is important to consider the current footprint of tools and APIs available in the industry for the standard.

8.2 Hardware Error Report Standardization Solutions

The proposed standardization consists of CPER format for the platform error logging with Redfish as the transport protocol.

- CPER based error Records - As CPER based Hardware errors are already standardized as part of the UEFI specification adopting this solution for out-of-band methods also gets a wider adoption.
- Redfish schema additions - Allow an individual LogEntry to reference a CPER Record or individual CPER Section as AdditionalData. Support can be easily provided using "CPER" to the list of supported formats for attached diagnostic data.
- Redfish "Platform Error" Message Registry – Define a set of messages to cover common hardware error cases associated with CPER data. The message registry provides a human-readable text message along with the programmatic means to identify specific errors without requiring text parsing.

As part of OCP and DMTF collaboration, error reporting standardization implementation details referenced here: https://www.dmtf.org/sites/default/files/Redfish_LogEntry_for_CPER_WIP.pdf.

8.3 GPU/Accelerators System level Errors to CPER/Redfish mapping

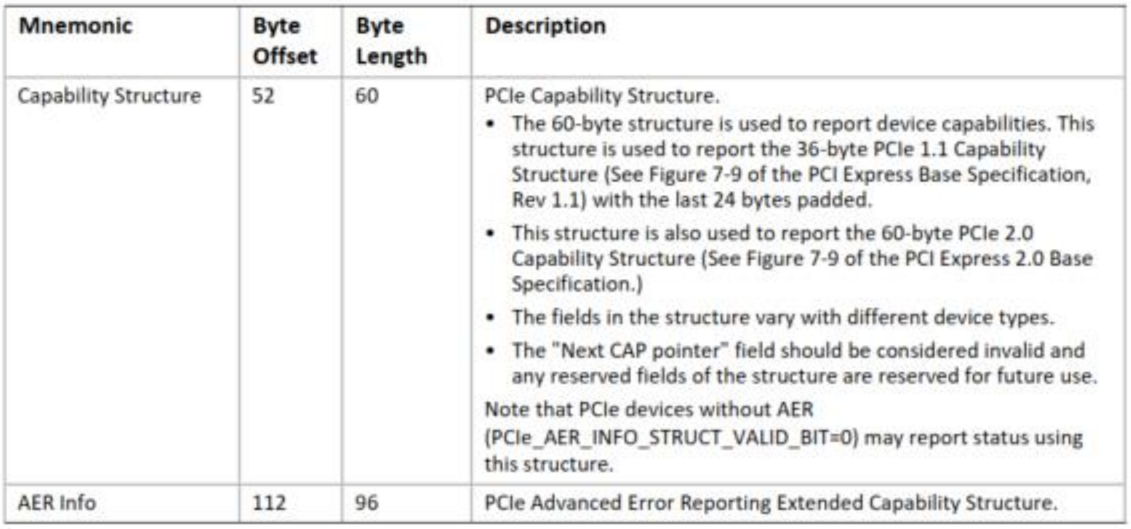
Error Type	**Reference of Error Record Format**	**Comments**
Memory Error	Section N.2.5 in UEFI spec Ver 2.9	9.2.2
PCIe Errors	Section N.2.7 in UEFI spec Ver 2.9	9.2.4
CXL Protocol Errors	Section N.2.13 in UEFI spec Ver 2.9	9.2.6
CXL Component Errors	Section N.2.14 in UEFI spec Ver 2.9	9.2.8
Platform Errors	Section N.2.3 in UEFI spec Ver 2.9	9.2.10Non-standard Section Body
	9.2.11	9.2.12

Example for CPER for PCIe Errors:

```
PCIe Advanced Error Reporting
              Corrected
------------------------------------------------------------
  0 - PCIe Error Section (Primary)
    Port Type : [4] Root Port
    Version   : 1.1
   Cmd/Status : 0x0010/0x0547
   Device ID  :
        VenId:DevId : 0x8086:0x352a
          Class Code: 0x030400
     Function Number: 0x00
       Device Number: 0x01
             Segment: 0x00
  Primary Bus Number: 0x15
Secondary Bus Number: 0x15
           Reserved1: 0x0
         Slot Number: 0x0000
           Reserved2: 0x00
Device Ser # : 0x0000000000000000
Bridge Ctl/Sts: not supplied
Exp Capability: 0x0142
Dev Caps      : 0x00008022
Dev Control   : 0x2127
Dev Status    : 0x0001
Link Caps     : 0x017a4105
Link Control  : 0x0040
Link Status   : 0xf101
Slot Caps     : 0x00080000
Slot Control  : 0x03c0
Slot Status   : 0x0040
Root Caps     : 0x0001
Root Control  : 0x0008
Root Status   : 0x00000000
Dev Caps2     : 0x007317f7
Dev Control2  : 0x0049
Dev Status2   : 0x0000
Uncorrectable Error Status   : 0x00000000
Uncorrectable Error Mask     : 0x00000000
Uncorrectable Error Severity : 0x00062010
Correctable Error Status     : 0x00000001
    Receiver Error
Correctable Error Mask       : 0x00000000
Caps & Control               : 0x000010a0
Header Log                   : 0x4a000001 0x16000004 0xfd000000 0x00000000
Root Error Command           : 0x00000007
Root Error Status            : 0x00000001
Correctable Err Source ID    : 0x15, 0x01, 0x00
Uncorrectable Err Source ID  : 0x00, 0x00, 0x00
```

Table N-33 PCI Express Error Record

Mnemonic	Byte Offset	Byte Length	Description
Validation Bits	0	8	Indicates which of the following fields is valid: Bit 0 –Port Type Valid Bit 1 – Version Valid Bit 2 – Command Status Valid Bit 3 – Device ID Valid Bit 4 – Device Serial Number Valid Bit 5 – Bridge Control Status Valid Bit 6 – Capability Structure Status Valid Bit 7 – AER Info Valid Bit 8-63 – Reserved
Port Type	8	4	PCIe Device/Port Type as defined in the PCI Express capabilities register: 0: PCI Express End Point 1: Legacy PCI End Point Device 4: Root Port 5: Upstream Switch Port 6: Downstream Switch Port 7: PCI Express to PCI/PCI-X Bridge 8: PCI/PCI-X to PCI Express Bridge 9: Root Complex Integrated Endpoint Device 10: Root Complex Event Collector
Version	12	4	PCIe Spec. version supported by the platform: Byte 0-1: PCIe Spec. Version Number • Byte0: Minor Version in BCD • Byte1: Major Version in BCD Byte2-3: Reserved
Command Status	16	4	Byte0-1: PCI Command Register Byte2-3: PCI Status Register
Reserved	20	4	Must be zero
Device ID	24	16	PCIe Root Port PCI/bridge PCI compatible device number and bus number information to uniquely identify the root port or bridge. Default values for both the bus numbers is zero. Byte 0-1: Vendor ID Byte 2-3: Device ID Byte 4-6: Class Code Byte 7: Function Number Byte 8: Device Number Byte 9-10: Segment Number Byte 11: Root Port/Bridge Primary Bus Number or device bus number Byte 12: Root Port/Bridge Secondary Bus Number Byte 13-14: Bit0:2: Reserved Bit3:15 Slot Number Byte 15 Reserved
Device Serial Number	40	8	Byte 0-3: PCIe Device Serial Number Lower DW Byte 4-7: PCIe Device Serial Number Upper DW
Bridge Control Status	48	4	This field is valid for bridges only. Byte 0-1: Bridge Secondary Status Register Byte 2-3: Bridge Control Register



Mnemonic	Byte Offset	Byte Length	Description
Capability Structure	52	60	PCIe Capability Structure. • The 60-byte structure is used to report device capabilities. This structure is used to report the 36-byte PCIe 1.1 Capability Structure (See Figure 7-9 of the PCI Express Base Specification, Rev 1.1) with the last 24 bytes padded. • This structure is also used to report the 60-byte PCIe 2.0 Capability Structure (See Figure 7-9 of the PCI Express 2.0 Base Specification.) • The fields in the structure vary with different device types. • The "Next CAP pointer" field should be considered invalid and any reserved fields of the structure are reserved for future use. Note that PCIe devices without AER (PCIe_AER_INFO_STRUCT_VALID_BIT=0) may report status using this structure.
AER Info	112	96	PCIe Advanced Error Reporting Extended Capability Structure.

8.3.1 Redfish Examples

LogEntry example - CPER with inline diagnostic data

```
{
"@odata.type": "#LogEntry.v1_14_0.LogEntry",
"Id": "3",
"Name": "CPER Log Entry with large additional data",
"EntryType": "Event",
"Severity": "Critical",
"Created": "2022-03-07T14:45:00Z",
"Message": “A platform error has occurred.",
"MessageId": "Platform.1.0.PlatformError",
"Links": {
"OriginOfCondition": {
"@odata.id": "/redfish/v1/Systems/1"
}
},
"CPER": {
"NotificationType": "902834BC-AD67-0BAD-BEEF-123456789012"
},
"DiagnosticDataType": "CPER",
```

“DiagnosticData”:

VGhlIGNha2UgaXMgYSBsaWUhCg==ASDEWIhnqn55Qe924MFAFHDFOIAFHEDANHV4582bAIYQN”,

"@odata.id": "/redfish/v1/Systems/1/LogServices/Log1/Entries/4",

}

9 References

- PCI Express Base Specification, Revision 4.0, September 27, 2017.
- Unified Extensible Firmware Interface Specification, version 2.9,

10 OCP Tenets

Openness

This specification has been developed via close and open collaboration between industry partners and competitors. Interface and Specification will be open to all OCP members.

Efficiency

The Goal of this specification is to make integration of GPUs into Hyperscaler solutions seamless. Companion compliance tooling will enable high quality products

Impact

This is the first Industry initiative to standardize GPU requirements. It is expected to have significant impact to quality and TTM for GPU Systems by Hyperscalers. Furthermore, it is expected to be applicable to Enterprise deployments as well.

Scale

Specification will apply to very large-scale GPU system deployments in Hyperscale Data Centers.

Sustainability

Reduce the component replacements rates through enhanced Availability and Fault Code improvements.

11 Appendices

11.1 Appendix A - Glossary of Terms

Term	Definition
Hyperscalers	Cloud service providers of the world who value Scaling and rapid time to market of new accelerator products.
GPU	A Graphics Processing Unit. It can be a discrete unit or something like a GPU card over PCIe.
GPU Accelerator	A GPU is not necessarily used not as a Graphics processing unit, but as a processing unit for some unit of work by hyperscalers.
SLO	
BMC Firmware	Firmware running on the BMC.
BMC-based error logging	Error logging using firmware running on the BMC, and not involving the operating system or platform firmware (may use management core firmware)
Containment	Error containment prevents corrupted data from being used by applications on the system and prevents corrupted data from being transmitted over I/O (e.g., stored to disk or sent over a network). In systems that had a bus-based architecture, resets could implement error containment since the reset was sent to all the devices on the bus at once. Modern systems do not use buses and the lack of a reset wire means there it takes a potentially long time for resets to propagate. Modern systems therefore transmit containment signals via their interconnect fabric (e.g., UPI or HyperTransport). See poison and viral, for example containment signals. Stopping an OS with an interrupt is not sufficient for containment. DMA may continue after the OS is interrupted and can send corrupted data to I/O.
CSP	Cloud Service Provider - The term Cloud Service Provider is intended to be a broad term for any organization that runs a large fleet of machines and provides services using that fleet. The use cases in this document are broad and are not

	strictly limited to organizations that provide cloud services. This proposal also supports use cases that include server operators that are not cloud providers.
Error Logging	Error logging is used very broadly to refer to collecting error data about a failure. Error logging is the process of latching information about an error in the chipset, collecting the error data in software and sending the data to tools where it can be analyzed. When software reads error logs, it often generates a binary structure such as a Common Platform Error Record (CPER) log that can be passed to other, higher-level software for analysis.
Error Handling	Error handling is where software logs errors, analyzes them, and determines how to minimize their impact. Error handling software can take actions like driving RAS features, telling the operating system to stop using resources and sending messages to get the server repaired.
Fault	A fault is something that causes hardware to malfunction. If the hardware detects the fault, it can log errors that aid in diagnosing the fault and servicing the hardware.
FIT	The Failures In Time (FIT) rate of a device is the number of failures expected in one billion (10^9) device-hours of operation. (E.g., 1000 devices for 1 million hours, or 1 million devices for 1000 hours each, or some other combination.)
FRU (Field Replaceable Unit	An FRU is a part that can be changed in the data center to repair a broken system. Components like DIMMs or PCIe cards are FRUs. A DRAM soldered to a mainboard would not be considered an FRU; in that case, the mainboard would be considered an FRU
Hardware component	A component designed by a vendor. Examples of hardware components include CPU chips/SoCs, DRAMs and PCIe devices.
Node	A platform designed by a vendor integrating multiple components that run an operating system. These are sometimes also called compute nodes.

Platform firmware	Firmware running on mission-mode cores on the processor. On x86 CPUs, platform firmware is sometimes called the BIOS or UEFI firmware.
Operating system (OS)	An operating system running on the processor. The operating system may run in a virtual machine on a hypervisor or directly on the machine (bare metal).
RAS Features	RAS features mitigate hardware faults. They can be implemented in hardware, software, or a combination of both. Some examples of RAS features are memory ECC, DRAM Post Package Repair, marking pages as bad in the OS and poison recovery. Some RAS features like Post Package Repair (PPR) need to be triggered. For example, on a memory controller that supports run-time PPR, software might see a series of corrected errors, all from the same row address. That software could decide that PPR would mitigate the fault causing the corrected row errors and could trigger hardware to do the PPR.
Signaling	Error signaling refers to how software is notified that an error has occurred.

11.2 Appendix B - Implementation Considerations

11.2.1 Memory Error Injection Schema

```
<!--  -->
<!--
##############################################################################
##        -->
<!-- # Redfish OEM Schema:  OcpMemory  v0.7.0
-->
<!-- #
-->
<!-- # Copyright 2023 Open Compute Project.
-->
<!-- # For the full OCP copyright policy, see LICENSE.md
-->
<!--
##############################################################################
##        -->
<!--  -->
<edmx:Edmx xmlns:edmx="http://docs.oasis-
open.org/odata/ns/edmx" Version="4.0">
```

```
<edmx:Reference Uri="http://docs.oasis-
open.org/odata/odata/v4.0/errata03/csd01/complete/vocabularies/Org.OData.Core.
V1.xml">
<edmx:Include Namespace="Org.OData.Core.V1" Alias="OData"/>
</edmx:Reference>
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/RedfishExtensions_v1.x
ml">
<edmx:Include Namespace="RedfishExtensions.v1_0_0" Alias="Redfish"/>
<edmx:Include Namespace="Validation.v1_0_0" Alias="Validation"/>
</edmx:Reference>
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Resource_v1.xml">
<edmx:Include Namespace="Resource"/>
</edmx:Reference>
<edmx:Reference Uri="http://docs.oasis-
open.org/odata/odata/v4.0/errata03/csd01/complete/vocabularies/Org.OData.Measu
res.V1.xml">
<edmx:Include Namespace="Org.OData.Measures.V1" Alias="Measures"/>
</edmx:Reference>
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Memory_v1.xml">
<edmx:Include Namespace="Memory.v1_0_0"/>
</edmx:Reference>
<edmx:DataServices>
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="OcpMemory">
<Annotation Term="Redfish.OwningEntity" String="OCP"/>
<Action Name="InjectCorrectableError" IsBound="true">
<Annotation Term="OData.Description" String="Injects a correctable error to a
specific persistent memory address in the memory device."/>
<Annotation Term="OData.LongDescription" String="This action shall inject a
correctable error to a specific persistent memory address in the memory
device."/>
<Parameter Name="Memory" Type="Memory.v1_0_0.Actions"/>
<Parameter Name="PhysicalAddress" Type="Edm.String" Nullable="false">
<Annotation Term="OData.Description" String="The device persistent physical
address in which to perform a correctable error injection as a hex-encoded
string."/>
<Annotation Term="OData.LongDescription" String="This parameter shall contain
the device persistent physical address in which to perform a correctable error
injection as a hex-encoded string."/>
<Annotation Term="Validation.Pattern" String="^0x[0-9a-fA-F]+$"/>
</Parameter>
</Action>
<Action Name="InjectUncorrectableError" IsBound="true">
<Annotation Term="OData.Description" String="Injects an uncorrectable error to
a specific persistent memory address in the memory device."/>
<Annotation Term="OData.LongDescription" String="This action shall inject an
uncorrectable error to a specific persistent memory address in the memory
device."/>
<Parameter Name="Memory" Type="Memory.v1_0_0.Actions"/>
<Parameter Name="PhysicalAddress" Type="Edm.String" Nullable="false">
<Annotation Term="OData.Description" String="The device persistent physical
address in which to perform an uncorrectable error injection as a hex-encoded
string."/>
<Annotation Term="OData.LongDescription" String="This parameter shall contain
the device persistent physical address in which to perform an uncorrectable
error injection as a hex-encoded string."/>
<Annotation Term="Validation.Pattern" String="^0x[0-9a-fA-F]+$"/>
</Parameter>
```

```
</Action>
</Schema>
<Schema xmlns="http://docs.oasis-
open.org/odata/ns/edm" Namespace="OcpMemory.v0_7_0">
<Annotation Term="Redfish.OwningEntity" String="OCP"/>
<Annotation Term="OData.Description" String="This version was created to add
the InjectCorrectableError and InjectUncorrectableError actions."/>
</Schema>
</edmx:DataServices>
</edmx:Edmx>
```

11.2.2 PCI-e Device Error Injection Schema

```
<!--  -->
<!--
################################################################################
##        -->
<!-- # Redfish OEM Schema:  OcpPCIeDevice  v0.7.0
-->
<!-- #
-->
<!-- # Copyright 2023 Open Compute Project.
-->
<!-- # For the full OCP copyright policy, see LICENSE.md
-->
<!--
################################################################################
##        -->
<!--  -->
<edmx:Edmx xmlns:edmx="http://docs.oasis-
open.org/odata/ns/edmx" Version="4.0">
<edmx:Reference Uri="http://docs.oasis-
open.org/odata/odata/v4.0/errata03/csd01/complete/vocabularies/Org.OData.Core.
V1.xml">
<edmx:Include Namespace="Org.OData.Core.V1" Alias="OData"/>
</edmx:Reference>
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/RedfishExtensions_v1.x
ml">
<edmx:Include Namespace="RedfishExtensions.v1_0_0" Alias="Redfish"/>
<edmx:Include Namespace="Validation.v1_0_0" Alias="Validation"/>
</edmx:Reference>
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Resource_v1.xml">
<edmx:Include Namespace="Resource"/>
</edmx:Reference>
<edmx:Reference Uri="http://docs.oasis-
open.org/odata/odata/v4.0/errata03/csd01/complete/vocabularies/Org.OData.Measu
res.V1.xml">
<edmx:Include Namespace="Org.OData.Measures.V1" Alias="Measures"/>
</edmx:Reference>
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/PCIeDevice_v1.xml">
<edmx:Include Namespace="PCIeDevice.v1_1_0"/>
</edmx:Reference>
<edmx:DataServices>
```

```
<Schema xmlns="http://docs.oasis-
open.org/odata/ns/edm" Namespace="OcpPCIeDevice">
<Annotation Term="Redfish.OwningEntity" String="OCP"/>
<Action Name="InjectCorrectableError" IsBound="true">
<Annotation Term="OData.Description" String="Injects a correctable PCIe
error."/>
<Annotation Term="OData.LongDescription" String="This action shall inject a
correctable PCIe error in the PCIe device."/>
<Parameter Name="PCIeDevice" Type="PCIeDevice.v1_1_0.Actions"/>
<Parameter Name="ErrorType" Type="OcpPCIeDevice.v0_7_0.CorrectableErrorType" N
ullable="false">
<Annotation Term="OData.Description" String="The error type in which to
perform a correctable PCIe error injection."/>
<Annotation Term="OData.LongDescription" String="This parameter shall contain
the error type in which to perform a correctable PCIe error injection."/>
</Parameter>
</Action>
<Action Name="InjectUncorrectableNonFatalError" IsBound="true">
<Annotation Term="OData.Description" String="Injects a non-fatal,
uncorrectable PCIe error."/>
<Annotation Term="OData.LongDescription" String="This action shall inject a
non-fatal, uncorrectable PCIe error in the PCIe device."/>
<Parameter Name="PCIeDevice" Type="PCIeDevice.v1_1_0.Actions"/>
<Parameter Name="ErrorType" Type="OcpPCIeDevice.v0_7_0.UncorrectableNonFatalEr
rorType" Nullable="false">
<Annotation Term="OData.Description" String="The error type in which to
perform a non-fatal, uncorrectable PCIe error injection."/>
<Annotation Term="OData.LongDescription" String="This parameter shall contain
the error type in which to perform a non-fatal, uncorrectable PCIe error
injection."/>
</Parameter>
</Action>
<Action Name="InjectUncorrectableFatalError" IsBound="true">
<Annotation Term="OData.Description" String="Injects a fatal, uncorrectable
PCIe error."/>
<Annotation Term="OData.LongDescription" String="This action shall inject a
fatal, uncorrectable PCIe error in the PCIe device."/>
<Parameter Name="PCIeDevice" Type="PCIeDevice.v1_1_0.Actions"/>
<Parameter Name="ErrorType" Type="OcpPCIeDevice.v0_7_0.UncorrectableFatalError
Type" Nullable="false">
<Annotation Term="OData.Description" String="The error type in which to
perform a fatal, uncorrectable PCIe error injection."/>
<Annotation Term="OData.LongDescription" String="This parameter shall contain
the error type in which to perform a fatal, uncorrectable PCIe error
injection."/>
</Parameter>
</Action>
</Schema>
<Schema xmlns="http://docs.oasis-
open.org/odata/ns/edm" Namespace="OcpPCIeDevice.v0_7_0">
<Annotation Term="Redfish.OwningEntity" String="OCP"/>
<Annotation Term="OData.Description" String="This version was created to add
the InjectCorrectableError, InjectUncorrectableNonFatalError and
InjectUncorrectableFatalError actions."/>
<EnumType Name="CorrectableErrorType">
<Member Name="ReceiverError">
```

```
<Annotation Term="OData.Description" String="This value shall indicate a
receiver error."/>
</Member>
<Member Name="BadTLP">
<Annotation Term="OData.Description" String="This value shall indicate a bad
transaction layer packet (TLP) error."/>
</Member>
<Member Name="BadDLLP">
<Annotation Term="OData.Description" String="This value shall indicate a bad
data link layer packet (DLLP) error."/>
</Member>
<Member Name="ReplayTimerTimeout">
<Annotation Term="OData.Description" String="This value shall indicate a
replay timer timeout error."/>
</Member>
<Member Name="ReplayNumRollover">
<Annotation Term="OData.Description" String="This value shall indicate a
REPLAY_NUM rollover error."/>
</Member>
</EnumType>
<EnumType Name="UncorrectableNonFatalErrorType">
<Member Name="PoisonedTLPReceived">
<Annotation Term="OData.Description" String="This value shall indicate a
poisoned transaction layer packet (TLP) received."/>
</Member>
<Member Name="ECRCCheckFailed">
<Annotation Term="OData.Description" String="This value shall indicate an ECRC
check failed."/>
</Member>
<Member Name="UnsupportedRequest">
<Annotation Term="OData.Description" String="This value shall indicate an
unsupported request error."/>
</Member>
<Member Name="CompletionTimeout">
<Annotation Term="OData.Description" String="This value shall indicate a
completion timeout error."/>
</Member>
<Member Name="CompleterAbort">
<Annotation Term="OData.Description" String="This value shall indicate a
completer abort error."/>
</Member>
<Member Name="UnexpectedCompletion">
<Annotation Term="OData.Description" String="This value shall indicate an
unexpected completion error."/>
</Member>
</EnumType>
<EnumType Name="UncorrectableFatalErrorType">
<Member Name="TrainingError">
<Annotation Term="OData.Description" String="This value shall indicate a
training error."/>
</Member>
<Member Name="DLLProtocolError">
<Annotation Term="OData.Description" String="This value shall indicate a data
link layer (DLL) protocol error."/>
</Member>
<Member Name="ReceiverOverflow">
```

```
<Annotation Term="OData.Description" String="This value shall indicate a
receiver overflow error."/>
</Member>
<Member Name="FlowControlProtocolError">
<Annotation Term="OData.Description" String="This value shall indicate a flow
control protocol error."/>
</Member>
<Member Name="MalformedTLP">
<Annotation Term="OData.Description" String="This value shall indicate a
malformed transaction layer packet (TLP) error."/>
</Member>
</EnumType>
</Schema>
</edmx:DataServices>
</edmx:Edmx>
```