OCP GPU & ACCELERATOR RAS REQUIREMENTS V0.9

Author (s):

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

[David Nieto, Drew Walton **Google**]

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

[Anil Agrawal, Akshat Nanda, Balaji Vembu **Meta**]

[Carlos Vallin, Vilas Sridharan **AMD**]

Table of Contents

Table of Contents 2

Table of Figures 3

Executive Summary 4

License 4

About Open Compute Foundation 5

1. Introduction 6

1.1 Goals 6

1.2 High level requirements to meet the goals: 6

1.3 GPU RAS System Level View 8

2. Problem Statement 10

3. Scope 12

4. Error Injection 13

4.1 Importance of Out of Band Error Injection 13

4.2 Error Injection Requirements 14

4.3 Error Coverage 14

4.4 Error Injection Attributes 15

4.5 Redfish-based Error Injection for GPU Accelerators 16

4.6 Device Identification using Redfish device tree 18

4.7 Memory Error Injection Details 19

4.8 PCIe Error Injection Details 20

4.9 Enabling Error Injection Mode 23

5. Error Reporting Standards 25

5.1 Requirements 26

5.2 Hardware Error Reporting Standardization 27

6. GPU and System debug dumps 27
6.1 System debug dump capabilities API: 28
6.2 System Level debug dumps 32
6.3 Debug dump tool analysis Requirements 34
6.4 Analysis Framework 35
7. RAS Feature Requirements 37
8. PCIe Fault Management and RCA Telemetry 37
9. GPU Fault Management software requirements 38
10. Compliance tools 39
11. References 39
12. OCP Tenets 40
Appendices 40
Appendix A – RAS Feature Requirements 40
Appendix B - Implementation Considerations 42
Appendix C - Glossary of Terms 49

Table of Figures

Figure 1: System Level View *8*
Figure 2: High-level flow for Redfish-based Error Injection *16*
Figure 3: Redfish Device Tree for the UBB Component *19*
Figure 4: Overview of Secure Unlocking *24*
Figure 5: Dump Record Format *34*
Figure 6: Overview of Dump Record Analysis *35*
Figure 7: Dump Record Analysis Process *35*

Executive Summary

With the explosion of AI/ML workloads, adoption of GPUs and Accelerators is rapidly accelerating in hyperscale Cloud Data Centers. To support fast adaptation and improved TTM for multiple accelerators from different vendors, we see a need to standardize some critical RAS (Reliability, Availability and Serviceability) capabilities. Some of the critical variances seen in the GPU and accelerator landscape include gaps in the validation tools (injecting hardware faults), Error reporting, debug dumps, and GPU fault management. Hyperscalers need the ability to exercise firmware and software stacks for RAS validation flows with hardware fault injections, to verify hardware error handling and resiliency flows, identify faulty FRU, RCA telemetry.

This document will cover RAS requirements, Redfish based standardization approaches towards hardware error injection, error reporting and debug dumps for different types of GPUs and accelerators. We propose 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 to mutually benefit both and propel our industry forward.

License

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

Google, Microsoft, NVIDIA, Meta, AMD

Usage of this Specification is governed by the terms and conditions set forth in **Open Web Foundation 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.

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.

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

1. Introduction

This document will focus on the Reliability, Availability and Serviceability (RAS) of GPUs and accelerators in the context of their usage with hyperscalers. Currently, there is no industry standard for exposing RAS capabilities and validating them. As a result, each GPU vendor has their own proprietary methods and resulting in barriers to adoption (requires Hyperscalers to implement bespoke tooling interfaces, high NRE for each new generation).

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 hyperscalers' function requirements, such that the adoption of their hardware into hyperscalers will be faster. In turn, suppliers will benefit by being able to seamlessly work with multiple hyperscalers in the future.

1.1 Goals

The goals of the RAS solution are:

1. Improved Node Uptime: The goal is to have a 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 Repair). This is achieved by identifying faulty components to the level of a Field Replaceable Unit (FRU).
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. Use scale hardware error data learnings and improve the quality of Silicon and reduce FIT rates.
5. Improve Time to Market (TTM): From a hyperscaler perspective, improve the time-to-market for existing and new accelerator hardware coming from suppliers.

1.2 High level requirements to meet the goals:

- **Fault-isolation to FRU level**

Devices shall report 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**

  Devices shall provide standardized hardware error injection tools which are compatible with cloud infrastructure to verify error detection, signaling, logging, and containment workflows.

- **Hardware Error resiliency**

  Minimizing the probability of a VM/Node crash due to hardware faults by detecting, correcting, signaling, and often containing errors rather than forcing system resets. Fault correction capabilities required are ECC, memory row-remapping (aka Post Package Repair – PPR), scrubbing, OS page-offline, link retry/retrain, and various other types of sparing capabilities. Signaling requirements include both in-band and out-of-band signaling. Error containment is required at various levels to reduce the blast radius due to an uncorrected error.

- **GPU system level crash dumps**

  In case of a fatal error, it is possible that UBB may crash. In such cases, there is a requirement to provide the crash dump with sufficient fault information to assist in locating the source of the crash and to perform root cause analysis (RCA). To meet hardware error RCA and SLA times, require GPU system level crash dumps with capabilities support, Triggering mechanisms and tools support to analyze dumps.

- **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.

- **GPU screening tools**

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

production environments.

- **Compliance Tools**

  Create open-source software that will behave like a compliance check for suppliers and system integrators to ensure GPU compliance with the RAS requirements described in this document.

1.3 GPU RAS System Level View

At a system level, GPU RAS features are implemented across multiple components as shown in the figure below.

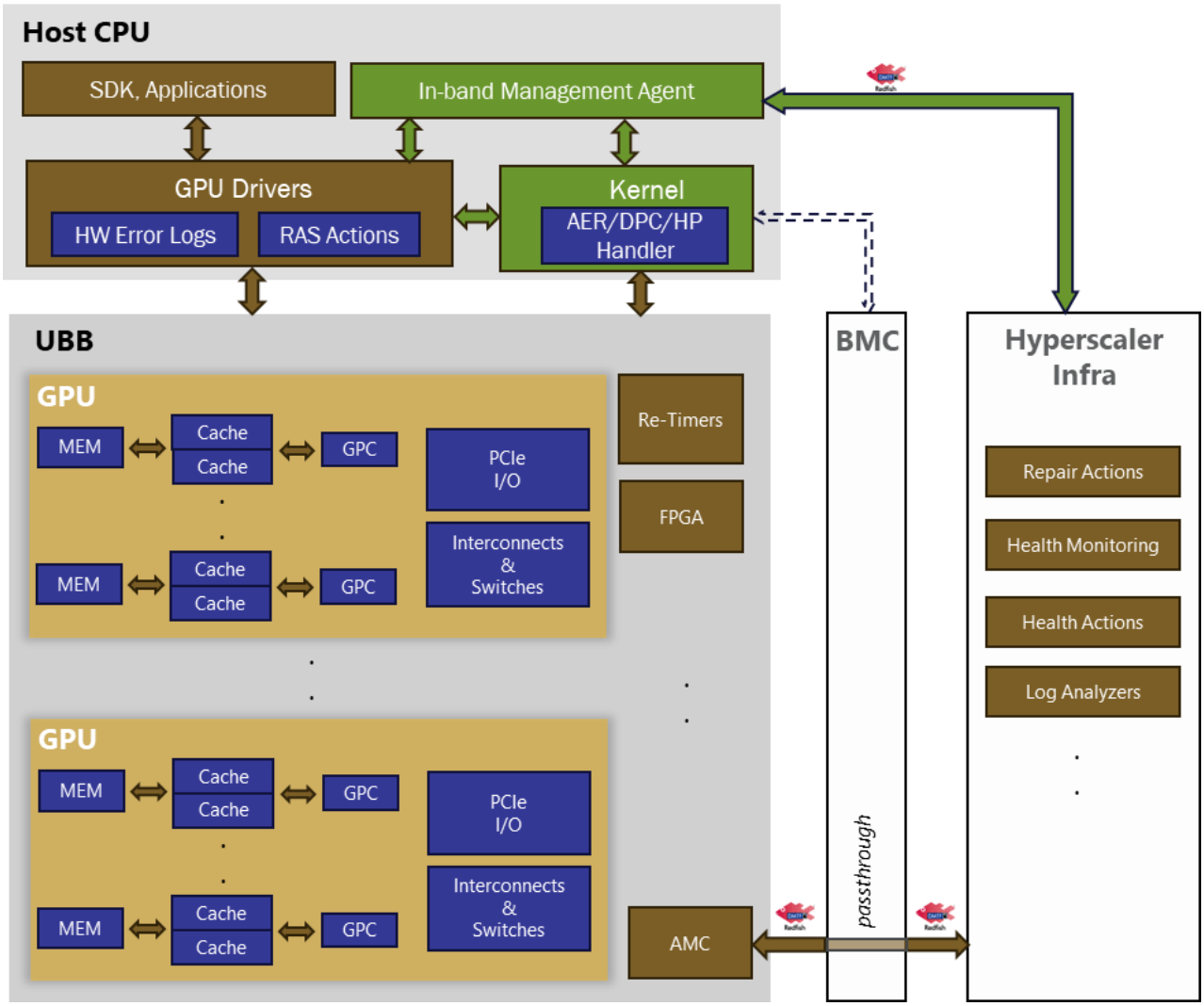

Figure 1: System Level View

- **MEM (Memory)**

  This includes the requirements such as memory error detection, correction, and reporting. In addition, system level features such as OS page offline, row-remapping, bad-page map out, and error containment are required.

- **Cache**

  This includes cache error detection, correction, and reporting. In addition, system level features such as bad-page map out, and error containment are required.

- **GPC**

  This includes various types of error detection, correction, and reporting within the computer core logic.

- **PCIe I/O**

  This includes PCIe specification defined error detection, correction, and reporting as part of 'Advanced Error Reporting' (AER), and Root Port Peripheral I/O (RP_PIO) error reporting. In addition, system level features such as 'downstream port containment (DPC)', 'Surprise Hot Plug' are required.

- **Interconnect and Switches**

  UBB designs have additional Switches and Interconnects, and support vendor specific error detection, correction, and reporting. Additionally, telemetry to detect link integrity issues is required.

- **Re-timers**

  Systems like UBB design have re-timers and require error collection related to PCIe link signal integrity and protocol errors and reporting either in-band and/or out-of-band.

- **Kernel (AER/DPC/HP Handler)**

  This includes Kernel components required to handle UBB specific failure events, such as AER handler, DPC handler, and Hot-plug handler. This also includes an interface to a device driver, and in-band management agent.

- **GPU Driver (Error Handler)**

  This includes the UBB specific error handlers that are managed via the driver, and various UBB vendor specific RAS actions.

- **AMC**

  This includes error collection and storage in persistent space for off-line error log collection via BMC and hyperscale infrastructure tools.

- **In-band Management Agent (IMA)**

This includes collecting error events in-band from UBB and presenting the logs to hyperscaler Infrastructure using redfish schema.

- **BMC for Out-of-band Management**
  This includes collecting error events out-of-band from UBB and presenting the logs to hyperscaler Infrastructure using redfish schema. It can also operate in pass-through mode where hyperscaler infrastructure can access the UBB telemetry data from the AMC.
- **Hyper-scale Infrastructure**
  This includes various levels of telemetry data aggregation, RAS action policy such auto-repair, or manual-repair, FRU isolation hints requirements for both in-band and out-of-band.

2. Problem Statement

Hyperscalers face challenges with the integration in accelerators, in part due to the software complexity and the heterogeneous nature of their implementation. One goal of this spec is to provide commonality in RAS implementation and interfaces (where appropriate) to reduce the burden of adopting new products. Examples of areas where commonality is required are: error logging, error reporting, error injection, and error analysis.

Additionally, accelerators must implement a certain feature set to be viable for hyperscalers. Another goal of this document is to define the minimum viable requirements for use in hyperscale data centers. Examples of areas where a minimum viable requirement is needed are: error detection/correction, and error analysis.

Typical pain points encountered by hyperscalers today have been summarized in the following table. Addressing these pain points would result in improving the Time to Market (TTM):

Example Problem	**Customer Workload Impact**	**Hyperscaler Requirement**
GPU not detectable on boot	Capacity Availability MTBF TTM Delays	Requirement: a reason code that specifies why the GPU was not detectable. Some possible causes may be: PCIe training issues Boot ROM corruption Bad Microcode update

Performance impact due to hardware errors	Workload Execution SLAs	Requirement: Sufficient visibility into hardware errors to understand that they are impacting performance. Possible causes may be: Error storms Link degradation Performance throttling due to thermal excursions or power problems Driver mishandling error conditions
GPU falling off the bus during run-time	Capacity Availability MTBF TTM Delays VM Crashes	Requirement: a reason code that specifies why the GPU is falling off the bus. Some possible causes may be: Uncontained ECC Errors (SRAM/DRAM) Row-remap ECC Failures PCIe Issue (link state, down, correctable, uncorrectable) Re-Timer failures
Silent Data corruption (SDC) due to hardware errors	Loss of customer confidence	Requirement: There is sufficient detection (in silicon, with test tools) to minimize the SDC rate. There is sufficient error containment for detected errors to prevent use of corrupted data. Possible causes include: Latent defects in silicon Soft errors Aging Lack of containment in the design
Uncorrectable errors due to hardware faults	Capacity Availability MTBF TTM Delays VM Crashes	**Requirement:** Fault management solution that minimizes downtime. **Includes:** Error logging and error analysis that drives RAS actions that minimize the impact of UCEs Error correction capabilities Error handling policies

Diagnose hardware failures not covered in row items above	Capacity Availability MTBF TTM Delays VM Crashes NFF replacements	**Requirement:** Fault management solution that enables hyperscalers to diagnose hardware failures and repair the failures. **Includes:** Telemetry adequate for diagnosing the problem (error logs and crashdumps) **Analysis tools to process telemetry:** Identifies the nature of the failure Identifies repair actions

3. Scope

This document aims to address the pain-points listed in section 2 by proposing standardization for GPU & Accelerators-based systems level Error Injection, Error Reporting, crash/Debug dumps. In addition it provides GPU & Accelerators-based systems RAS feature requirements, GPU and PCIe fault management software requirements.

The Error injection solution presented in this version of the requirements document focuses on OOB using AMC-based redfish interfaces and initial focus on error injection to support system during the development effort. In the future versions of the document will be amended to require error injection capabilities on production quality hardware.

Error reporting uses CPER based error record for out-of-band Error reporting. Error reporting and mitigation aspects of this specification must apply to production level hardware and firmware.

In addition, the current version of document provides high level requirements for GPU & Accelerators-based systems RAS feature requirements, GPU and PCIe fault management software requirements. Next revisions of the document provide detailed requirements.

4. Error Injection

4.1 Importance of Out of Band Error Injection

Typically, error handling and recovery code-paths on GPUs are not tested thoroughly enough in hyperscalar environments. Natural fault-occurrences cannot be relied upon for validation, so hyperscalers need the ability to induce errors on-demand.
GPU systems have complex topologies with multiple components in their path, like switches, re-timers, GPUs, CPUs, etc., so Intra-node RAS validation is critical. Additionally, fleet-level end-to-end error propagation and containment flows are also critical for hyperscalers to exercise to validate mitigation/repairability workflows.

For example, hyperscalers often encounter GPUs dropping off the PCIe bus. There could be various contributing factors to this, ranging from the GPU core to the PCIe devices in the path. Identifying the root-cause and potential contribution of GPUs to this issue is essential. Error-injection allows hyperscalers to verify their designs and characterize such failures along with identifying their true nature. For instance, GPU framing errors frequently masquerade themselves as correctable errors in intermediate PCIe switches. Error injections allows testing of the end-to-end error propagation behavior and understanding the impact to the system and workloads.

Currently, GPU fault handling is opaque to hyperscalers. When a fault occurs, it is difficult to know exactly what information to collect to analyze the failure. Even when collecting information, it may not be possible to disambiguate the data to deduce the remediation information needed for repairability workflows.
Furthermore, hyperscalers often implement custom host OS stacks, making it critical to have the ability to inject errors to verify fault-handling mechanisms in these implementations.

By standardizing error injection and verification methodologies, hyperscalers can obtain fault information form GPUs more efficiently, allowing the hyperscalers to move faster and improve their Time-To-Market (TTM). Similarly, suppliers may benefit from consistent interfaces against which they can implement their technology (e.g., re-timer devices, switches, etc.)

Catastrophic errors encountered by GPUs are sent to the host SW stack but are only visible to the guest, making this information mostly unusable to hyperscalers. It is critical for hyperscalers to have access to this

type of error information via an out-of-band methodology to enable them to intervene and remediate the issue without end-customer dependencies.

4.2 Error Injection Requirements

1. Error injection methods shall provide the following for verification:
    a. Error signaling – Error injection methodology should make every effort to exercise the signaling flows of naturally-occurring errors, instead of just simulating the error without invoking corresponding signaling flows.
    b. Faulty FRU identification.
    c. Verification of the Error Mitigation.
    d. Verification of the error flow handling in the platform hardware, firmware, and SW stack.
2. Error injection capability to support different HW error types at the IP Level, SOC Level, and on the platform level (e.g., HBM, PCIe).
3. Error Injection capability to specify the error severity: Uncorrectable, Correctable, and Fatal.
4. Error injection methods and interfaces shall be abstracted from silicon specific implementations from hyperscaler point-of-view.
5. Error injection methodologies should provide a secure mechanism to enable/disable error injection mode in production-grade devices.
6. Errors listed in this specification shall be injected solely using the Redfish schema defined in this specification.

4.3 Error Coverage

This version covers memory and PCIe error injections. Details are as follow:

Error type	Details	Summary/Comments
Memory errors	Device Memory (HBM, GDDR)	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	Where applicable, 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.

The future version of this document will include the following error types:

Error type	Details
GPU Core Errors	Internal GPU micro controller exceptions, GPU SRAM, Cache and firmware faults
GPU Links and Switches	GPU Interconnects, Interconnect Switches
Platform Specific Errors	FPGA Specific Errors, Thermal, USB VNIC, I2C, I3C

4.4 Error Injection Attributes

PCIe Errors Attributes

Attributes	Details	Comments
Device Identification	Uses Redfish based URL	Specifies the Device targeted for error injection
Error Severity	PCIe Correctable PCIe Non-Fatal and PCIe Fatal	All PCIe error severities shall 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 applicable error severity need to be supported

Memory Errors Attributes	Details	Comments
Physical Device Identification	Redfish-based URI	Specifies the Device targeted for error injection
Sub-Device Identification	Rank, Column, Row Level	Specifies the sub-device level targeted for error injection
Error Severity	Correctable Uncorrectable	All error severities shall be supported
Address	Memory Physical Address targeted for error injection	Optional if using Sub-Device Identification attribute

Note: Memory Error Poison and other errors are planned to be defined in the next revision of this document

4.5 Redfish-based Error Injection for GPU Accelerators

This section provides details about error injection at various hardware component levels. The initial focus of this document is to provide specific resource examples for Memory and PCIe, which can be expanded to other resources that the GPU/Accelerator vendors can support as OEM extensions. Errors shall be injected via Redfish actions.

The following diagram provides a high-level flow for Redfish-based error injection.

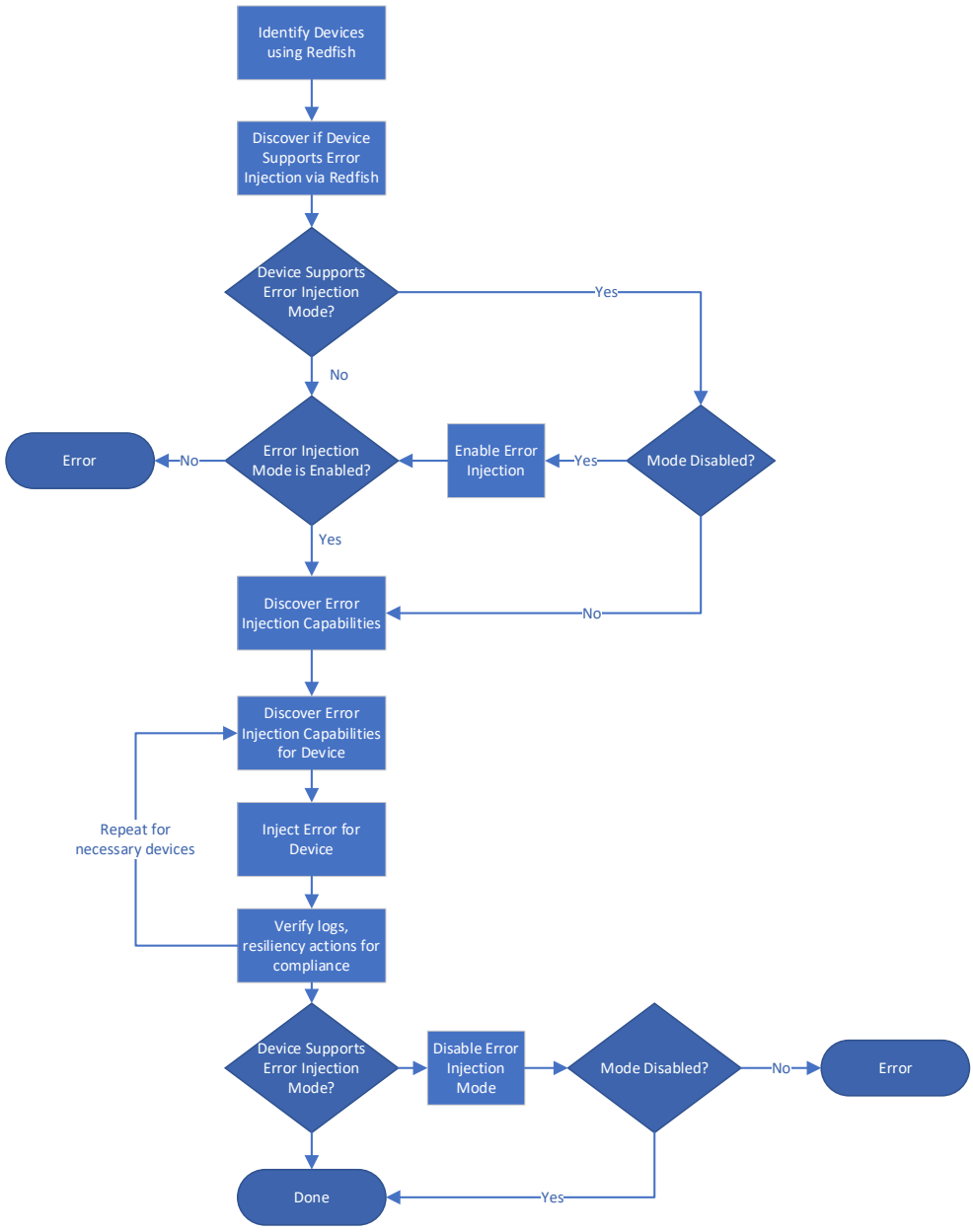

Figure 2: High-level flow for Redfish-based Error Injection

1. Device Identification – Using Redish Device Tree
2. Discover error injection capabilities, including methods supported to enable error injection mode. The discovery mechanism shall be covered in future revisions of this spec.
3. Enable error injection mode. Refer to section 4.9 for more details.
4. Perform error injection to a particular Device.
5. Compliance - Verify the Error logs and Error resiliency Actions.
6. Disable error injection mode, if desired.

Redfish API shall be utilized for supporting this flow. For instance, performing a "GET" on the resource provides supported error injection actions:

- Memory error injection actions are available in the Memory resource. The URI format shall follow the patterns defined in the Memory schema.
- PCIe error injection actions are available in the PCIe device resource. The URI format shall follow the patterns defined in the PCIe device schema.

Redfish Clients shall traverse to the GPU resource, then do a "GET" on the Memory or the PCIe device resources. The response includes error injection action details in the "Links" object. Example flow:

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>"
    },
    ...
}
```

b. "GET" on a Memory resource:

```
{
  "@odata.id": "<Memory ResourceUri> ",
  "Actions": {
     "Oem": {
```

```
        "#OCPMemory.InjectCorrectableECCErrors": {
          "target": "<Memory Resource
Uri>/Actions/Oem/OCPMemory.InjectCorrectableECCErrors",
          "@Redfish.ActionInfo": " <Memory
ResourceUri>/Oem/OCP/InjectCorrectableECCErrorActionInfo"
        },
        "#OCPMemory.InjectUncorrectableECCErrors": {
          "target": <Memory
ResourceUri>/Actions/Oem/OCPMemory.InjectUncorrectableECCErrors",
          "@Redfish.ActionInfo": " <Memory
ResourceUri>/Oem/OCP/InjectUncorrectableECCErrorActionInfo"
        }
      }
  }
  ...
}
```

c. "POST" on action target URI injects the desired error:

`POST {ResourceUri}/Oem/Actions/{ResourceType}.{ActionName}`

Memory error injection examples:

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

```
POST {Memory ResourceUri}/Actions/Memory.InjectPersistantPoison HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
  "PhysicalAddress": "0x1000"
}
```

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

```
POST {Memory
ResourceUri}/Actions/Oem/OCPMemory.InjectUncorrectableECCError HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
  "PhysicalAddress": "0x1000"
}
```

d. "GET" on a metric resource (e.g., "MemoryMetrics") will reflect an updated value in the error counters. For example, the value of "UncorrectableECCErrorCount" property shall be increased after a successful uncorrectable error injection.

e. In some cases, a corresponding error log entry may be created in the LogService resource.

4.6 Device Identification using Redfish device tree

The following figure provides an example of a Redfish device tree for the UBB component:

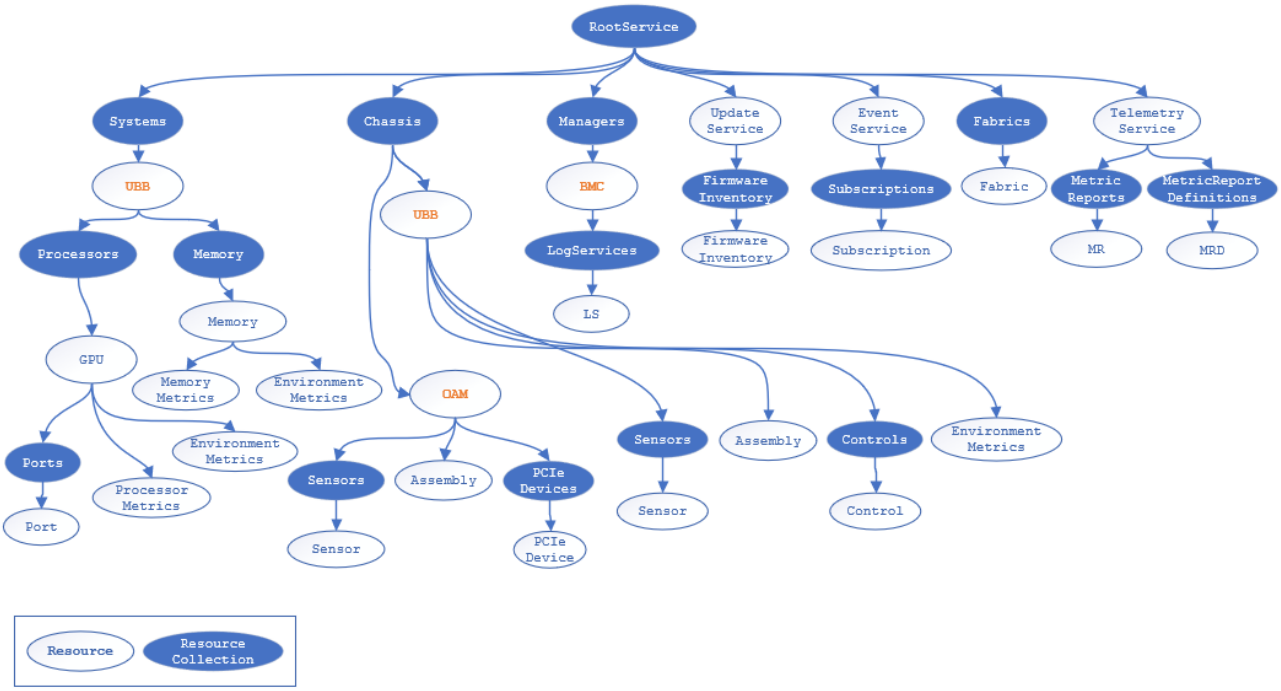

Figure 3: *Redfish Device Tree for the UBB Component*

Please refer to the [OCP GPU & Accelerator Management] for the latest on this.

4.7 Memory Error Injection Details

This document proposes the creation of two new OEM OCP actions: `InjectCorrectableECCError` and `InjectUncorrectableECCError`. The following table provides more details.

Property Name	Schema(s)	Type	Description
`InjectCorrectableECCError`	OCPMemory(Actions)	Object	Injects correctable ECC errors to the targeted memory device.
`InjectUncorrectableECCError`	OCPMemory(Actions)	Object	Injects uncorrectable ECC errors to the targeted memory device.

The following tables provide more details about each property.

InjectCorrectableECCError & InjectUncorrectableECCError actions in OCPMemory schema:

```
"InjectCorrectableECCError": {
    "description": "Injects a correctable ECC error to a specific memory address
in the memory device. ",
    "parameters": {
        "PhysicalAddress": {
```

```
            "description": "The device physical address as a hex-encoded string.",
            "requiredParameter": true,
            "type": "string"
        }
    }
},
"InjectUncorrectableECCError": {
    "description": "Injects an uncorrectable ECC error to a specific 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": {
        "#OCPMemory.InjectCorrectableECCErrors": {
          "target": "<Memory
ResourceUri>/Actions/Oem/OCPMemory.InjectCorrectableECCErrors",
          "@Redfish.ActionInfo": "<Memory
ResourceUri>/Oem/OCP/InjectCorrectableECCErrorActionInfo"
        },
        ‟#OCPMemory.InjectUncorrectableECCErrors": {
          "target": "<Memory
ResourceUri>/Actions/Oem/OCPMemory.InjectUncorrectableECCErrors",
          "@Redfish.ActionInfo": "<Memory
ResourceUri>/Oem/OCP/InjectUncorrectableECCErrorActionInfo"
        }
     }
  }
  …
}
```

4.8 PCIe Error Injection Details

This section details the proposed schemas to support Redfish-based error injection actions in PCIe 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

The following tables provide more details about each property.

```
"InjectCorrectableECCError": {
    "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”
           }
    }
}
```

```
        …
"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":{
        "#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"
        }
     }
  }
  ...
}
```

4.9 Enabling Error Injection Mode

As illustrated in Figure 4, error injection mode may need to be enabled before errors can be injected. The supplier may have disabled error injection by default in production-grade parts with security in mind, thus requiring the GPU accelerator (discrete or otherwise) to be put into a mode that allows it to accept error injection requests. This can be achieved through various authorization mechanisms, including unlocking the component, or via alternate authorization mechanisms advertised during the discovery phase (#2 in diagram). This section details the requirements and mechanisms to support this process.

Once the GPU accelerator is ready to accept error injection requests via Redfish, one can request areas of supported error injection from the GPU accelerator (step #3). Using this information, one can continuously inject errors in a loop, and receive telemetry confirmation and mitigation of the injected errors (step #4).

Finally, one should be able to disable error injection mode. This can be achieved through various mechanisms, including re-flashing the accelerator with production-grade firmware, resetting its security policy, or via the alternate authentication mechanism advertised in the discovery phase.

Suppliers must provide hyperscalers with “development” grade or “debug” grade builds/cards of GPU accelerators. Such hardware and firmware that is running on it shall have error injection enabled by default and support the error injection requirements specified in this document.

Production-grade builds/cards shall have error injection disabled by default. There shall be a mechanism to enable and disable error injection. Two methods are discussed in this document revision:

4.9.1 Enable Error Injection by Secure Unlocking

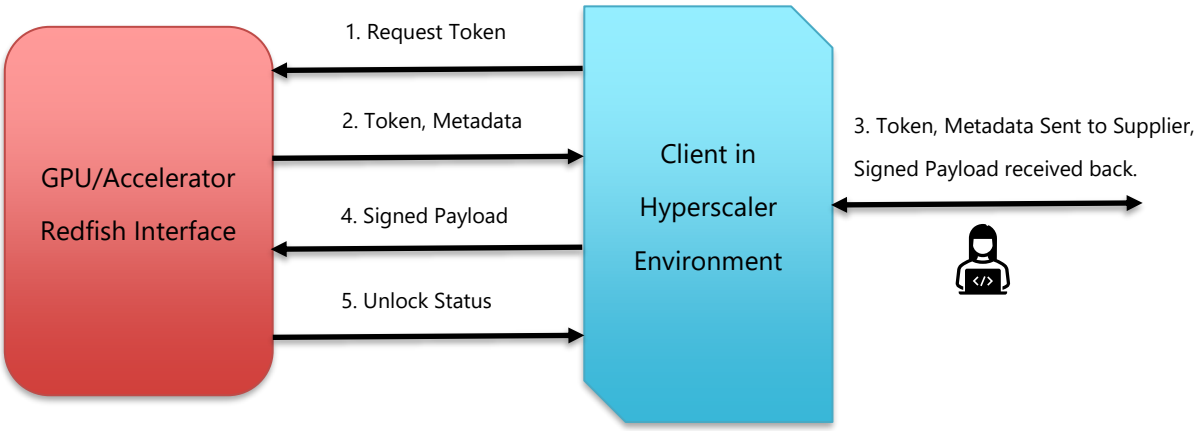

Figure 4: *Overview of Secure Unlocking*

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.

5. Hyperscaler (e.g., via host BMC/cloud agent) requests an unlock token from the accelerator complex (e.g., accelerator or AMC).
6. Hyperscaler receives a device-specific payload from the accelerator, for example a token + metadata.
7. Through an out-of-band process, the hyperscalers have the supplier process the payload and receive an unlock payload to send to the accelerator to unlock it. Definition of this out-of-band process is outside the scope of this document.
8. 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 not any other GPU accelerator hardware.
9. The accelerator unlocks and sends status.

After the Hyperscaler completes its testing cycles, it may need to re-lock the accelerator. One way to re-lock unlocked production-grade hardware is to re-flash it with production-grade firmware. Another way is to issue

appropriate command(s) to the point of authorization (BMC or accelerator complex) to end the error injection session or reset the security policy. Cases that require firmware re-flashing should be done via a standard firmware update process as specified in the FW Update Specification.

Note that the above unlocking mechanism is expected to be used as a one-off on a case-by-case basis. It is not intended to be used in production or in the factory during manufacturing.

To maintain a consistent abstraction from the Hyperscaler infrastructure across different vendors and platforms, future revisions of this document will propose a schema to standardize this process.

4.9.2 Enable Error Injection by Alternate Authorization Mechanism

Requiring a full unlock of the accelerator or contacting the supplier to enable error injection may not be feasible for some use cases. Therefore, an alternate authorization mechanism, such as using pre-provisioned Hyperscaler-owned keys or certificates in the platform, may be used to authenticate the request to enable error injection mode. Actual implementation may be vendor-specific to meet appropriate product-level threat models, as long as it meets the following requirements:

- Error Injection authorization and its required attributes are discoverable via standard Redfish schema discovery mechanisms.
- The authorization mechanism implemented must be part-specific; only applicable to the device it is requested for and does not include any other GPU accelerator hardware.
- Hyperscaler can request error injection mode actions (enable, disable) and query error injection mode status via Redfish API.
- Must include a mechanism to notify Hyperscaler infrastructure that error injection mode has been enabled on production grade hardware and firmware.
- Scalable solution that ensures error injection can be enabled in Hyperscaler environment.

To maintain a consistent abstraction from the Hyperscaler infrastructure across different vendors and platforms, future revisions of this document will propose a schema to standardize this process.

5. Error Reporting Standards

Current hardware error reporting formats used on hyperscaler deployments lack standardization and vary across CSPs. Additionally, hardware errors can be reported by different agents, such as UEFI Firmware, a

System Management Controller (e.g., BMC) and the Operating System. Standardizing error formats across different reporting agents can simplify error log harvesting tools and debug efforts.

There are two existing Industry Standards relevant to hardware error reporting in this document's scope. The first one is the IPMI-based System Event Logs (SEL). However, IPMI SEL has some limitations including reporting very primitive data for hardware errors, lack of support for new technologies such as CXL, and limited industry maintenance. The alternative is Redfish, which better caters to modern datacenter environments. While Redfish does not directly define hardware error formats, it provides an infrastructure upon which retrieval of hardware error records can be standardized.

The second Industry Standard relevant to hardware error reporting is the Common Platform Error Record (CPER) defined in the Unified Extensible Firmware Interface (UEFI) specification. UEFI firmware implementations use CPER records to share hardware error information with the OS. However, CPER records are not commonly used when accessing hardware error information out-of-band through the BMC, which is one of the most widely used mechanisms to harvest error data reliably at scale in both non bare-metal and bare-metal use cases.

This emphasizes the need for hardware error record format standardization, and the APIs to access these records. The error reporting standards selected must enable accelerated deployment of heterogenous hardware with minimal implementation and maintenance costs.

5.1 Requirements

Goals for hardware error reporting format:

- Common (vendor- and device-agnostic) format, usable across different error reporting agents.
- Extensible structure, to maximize its useful life and scale to future technologies or methodologies.
- Parseable by automation tools.
- Identify the faulty Field Replaceable Unit (FRU), and any information necessary to fully categorize the error.
- Provide additional diagnosis information to assist in root cause analysis, which may expand beyond hardware data.

Goals for hardware error logging methodologies:

- Self-contained (hermetic) and does not require additional information to be harvested by the BMC or data center telemetry services.
- Versioned, to allow updates to the format.
- Report errors to a node's Baseband Management Controller (BMC) as the point of aggregation for all error records.

5.2 Hardware Error Reporting Standardization

This document proposes using a combination of CPER as the hardware error record format and Redfish as the standardized transport protocol. The requirement per this specification is that GPU and Accelerators report error logs in CPER format over Redfish, specifically through Redfish LogEntry and/or Event resources. As part of OCP and DMTF collaboration, the details to implement CPER record reporting through Redfish are referenced here: https://www.dmtf.org/sites/default/files/Redfish_LogEntry_for_CPER_WIP.pdf.

A future revision of this specification will also define a Redfish "Platform Error" message registry, which defines a set of messages to cover common hardware error cases. This is intended to provide human-readable text along with machine-parseable logs.

6. GPU and System debug dumps

At hyperscale, hardware faults are inevitable, but how quickly we can RCA and apply the mitigations is critical to reduce the impact of these faults on the customer experience. Note that this specification does not dictate how these requirements may be implemented by the suppliers.

Fault isolation telemetry requires different levels of detail which depend on the type of hardware failure.

Error details	What it provides	Fleet actions	Comments/Example
Level 1	What Error happened	We do not know details about the error or how to RCA the issue	Limited to category and nature of error. Example: Error type: PCIe Uncorrectable errors

Level 2	Details about the error using CPER	We have full details about the error, which supports RCA efforts.	Example: PCIe Error full AER register data in CPER format
Level 3	Debug log dumps – Useful for system level fatal error RCA	Required in complex scenarios	Crash dumps or debug logs and System configuration and environmental data (e.g., Temperature)

6.1 System debug dump capabilities API:

System debug dump capabilities will vary based on the system design based on the GPU UBB and Discrete accelerator device. Need to support the Redfish based API to list the system debug dump capabilities. System debug dump capability API must return the following information.

Application codes traverse all `LogService` URIs defined in the `LogService` schema (https://redfish.dmtf.org/schemas/v1/LogService_v1.xml).. Then, do a GET on the "`@Redfish.ActionInfo`" value in the `CollectDiagnosticData` action object.

This is an example of the flow for discovering `LogService` subordinate to a `Manager` resource:

1. In a service root object `/redfish/v1`, look for a property called "`Managers`"
    a. Find the "`@odata.id`" value. This is the URI of the Manager collection.
    b. Do a `GET` on that URI.
2. Look at the "`Links`" object at the "`Members`" array.
    a. Find the "`@odata.id`" of the Manager in question (this may require GETs on each manager and looking at properties in the system to determine the right manager).
    b. Do a `GET` on that URI.
3. Look for an object called "`LogServices`".
    a. Find the "`@odata.id`" value.
    b. Do a `GET` on that URI.
4. Look at the "`Links`" object at the "`Members`" array.
        i. Find the "`@odata.id`" of the "`LogServices`" in question (this may require GETs on each LogService and looking at properties in the LogService to determine the right log service).

ii. Do a GET on that URI.

5. Look in the “Actions” object for an object called "#LogService.CollectDiagnosticData"
    a. Find the "@Redfish.ActionInfo" value.
    b. Do a GET on that URI – this provides the allowable values of the CollectDiagnosticData action.

Example response of a LogService resource:


```
{
    "@odata.id": "/redfish/v1/Managers/1/LogServices/Log1",
    "@odata.type": "#LogService.v1_5_1.LogService",
    "Id": "Log1",
    "Name": "Manager Log Service",
    "MaxNumberOfRecords": 1000,
    "OverWritePolicy": "WrapsWhenFull",
    "DateTime": "2015-03-13T04:14:33+06:00",
    "DateTimeLocalOffset": "+06:00",
    "ServiceEnabled": true,
    "LogEntryType": "Event",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "Actions": {
        "#LogService.CollectDiagnosticData": {
            "target":
"/redfish/v1/Managers/1/LogServices/Log1/Actions/LogService.CollectDiagnos
ticData",
            "@Redfish.ActionInfo":
"/redfish/v1/Managers/1/LogServices/Log1/CollectDiagnosticDataActionInfo"
        }
    },
    "Entries": {
        "@odata.id": "/redfish/v1/Managers/1/LogServices/Log1/Entries"
    }
}
```


Example response of an ActionInfo resource:


```
{
```

```
   "@odata.id":
"/redfish/v1/Managers/{ManagerId}/LogServices/{LogServiceId}/CollectDiagno
sticDataActionInfo,
   "@odata.type": "#ActionInfo.v1_0_0.ActionInfo",
   "Id": "CollectDiagnosticDataActionInfo",
   "Name": "Collect Diagnostic Data Action Info",
   "Parameters": [{
      "Name": "DiagnosticDataType",
      "Required": true,
      "DataType": "String",
      "AllowableValues": ["Manager"]
   }]
}
```

6. For each entry, a URL with details about information and tools to parse the dump.
7. For each entry, list of triggers i.e. if the dump collection is triggered automatically (on error) or needs to be requested explicitly by the user.
8. Estimated maximum size details will be a part of the next revision of the spec.
9. Estimated max time taken for the dump to be generated.

The API can be an enhancement over the following API (currently covers only item#1 above)

`ActionInfo` URI:

- `/redfish/v1/Managers/{ManagerId}/LogServices/{LogServiceId}/CollectDiagnosticDataActionInfo`
- `/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId}/CollectDiagnosticDataActionInfo`

Return Payload:

`/redfish/v1/Managers/{ManagerId}/LogServices/{LogServiceId}/CollectDiagnosticDataActionInfo`

```
{
   "@odata.id":
"/redfish/v1/Managers/{ManagerId}/LogServices/{LogServiceId}/CollectDiagnosticDa
taActionInfo,
   "@odata.type": "#ActionInfo.v1_0_0.ActionInfo",
   "Id": "CollectDiagnosticDataActionInfo",
   "Name": "Collect Diagnostic Data Action Info",
   "Parameters": [{
      "Name": "DiagnosticDataType",
```

```
      "Required": true,
      "DataType": "String",
      "AllowableValues": ["Manager"],
      "AllowableValueDescriptions": ["Tarball of all OS level logs on the AMC
(syslog, journalctl, etc.) "],
      " Oem ": {
            "OCP": {
                  "@odata.type": "#OCPActionInfo.v1_0_0.OCPActionInfo",
                  "AllowableValueDetails": [
                        {
                              "Value": "Manager",
                              "EstimatedDuration": "PT1H"
                        }]
                  }
            }
   }]
}
```

Return Payload:

```
/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId}/CollectDiagnos
ticDataActionInfo
```

```
{
   "@odata.id":
"/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId}/CollectDiagno
sticDataActionInfo
",
   "@odata.type": "#ActionInfo.v1_0_0.ActionInfo",
   "Id": "CollectDiagnosticDataActionInfo",
   "Name": "Collect Diagnostic Data Action Info",
   "Parameters": [{
      "Name": "DiagnosticDataType",
      "Required": true,
      "DataType": "String",
      "AllowableValues": ["OEM"]
   },{
      "Name": "OEMDiagnosticDataType",
      "Required": true,
      "DataType": "String",
      "AllowableValues": ["<OemDiagnosticType_1>", "<OemDiagnosticType_2>",
...],
      "AllowableValueDescriptions": ["Tarball of OemDiagnosticType_1 logs",
"Tarball of OemDiagnosticType_2 logs", ...],
```

```
        "Oem": {
            "OCP": {
                "@odata.type": "#OCPActionInfo.v1_0_0.OCPActionInfo",
                "AllowableValueDetails": [
                    {
                      "Value": "<OemDiagnosticType_1>",
                      "EstimatedDuration": "PT1H"
                    },{
                      "Value": "<OemDiagnosticType_2>",
                      "EstimatedDuration": "PT1H"
                    }]
                }
            }
         }
    }]
}
```

6.2 System Level debug dumps

To reduce fleet issue RCA SLAs, support for secure debug dumps and analyzer tools are critical. Certain hardware bugs only manifest in Hyperscale production fleet environments when exposed to customer workloads. The time to reproduce similar issues in debug or quarantine pools is non-deterministic and could take weeks to months. Collecting system level debug dumps at the time of failure is critical.

As you see GPU RAS System level view in section 5, GPU sub-systems have multiple components and require multiple debug dump collections and analyzer tools to process these dumps and drive repair actions.

Errors can be at System wide crash and Failure at Component level. The table below lists dump details, collection triggers and API support.

Dump type	**Dump Details**	**Dump collection triggers**	**Redfish API Support**
Manager level dump	AMC OS level logs to provide health of AMC	Manual	`curl -X POST http://${hmc}/redfish/v1/Managers/{ManagerId}/LogServices/Dump/Actions/LogService.CollectDiagnosticData -d '{"DiagnosticDataType": "Manager"}'`
System wide	self-test dumps Results of sanity health checks for System	Manual on demand, automatically on-error	`curl -X POST http://${ManagerIP}/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId}/Actions/LogService.CollectDiagnosticData/ -d '{"DiagnosticDataType":"OEM", "OEMDiagnosticDataType":"DiagnosticType=SelfTest"}'`

Component Level dump	Component Vendor defined	Manual on demand, automatically on-error	`curl -X POST http://${HMC_IP}/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId}/Actions/LogService.CollectDiagnosticData/ -d '{"DiagnosticDataType":"OEM", "OEMDiagnosticDataType":"DiagnosticType=<ComponentID>"}'`

Component Level debug dump types

Component level debug dumps can vary from platform. The elow table provides specific examples for a generic UBB design

Dump type	Coverage	Dump collection triggers	Redfish API Support
PCIe Retimer	LTSSM , FoM, (Figure of Merit)	Manual on demand	`curl -X POST http://${ManagerIP}/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId} /Actions/LogService.CollectDiagnosticData/ -d '{"DiagnosticDataType":"OEM", "OEMDiagnosticDataType":"DiagnosticType=RetLTSSM"}'`

GPU debug dump	GPC, Cache HBM	Manual on demand	`NA (Available in-band only)`
			
			`curl -X POST http://${ManagerIP}/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId} /Actions/LogService.CollectDiagnosticData/ -d '{"DiagnosticDataType":"OEM", "OEMDiagnosticDataType":"DiagnosticType=GPU"}'`

FPGA	Register Dumps	Manual on demand	`curl -X POST http://${ManagerIP}/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId} /Actions/LogService.CollectDiagnosticData/ -d '{"DiagnosticDataType":"OEM", "OEMDiagnosticDataType":"DiagnosticType=FPGA"}'`
EROT	PDL logs	Manual on demand	`curl -X POST http://${ManagerIP}/redfish/v1/Systems/{ComputerSystemId}/LogServices/{LogServiceId}/Actions/LogService.CollectDiagnosticData/ -d '{"DiagnosticDataType":"OEM", "OEMDiagnosticDataType":"DiagnosticType=EROT"}'`

Future spec revisions of the documents will further cover support for PCIe Switch, X-link Switch, GPU Debug Dump for GPC, Cache and HBM related and other components.

6.2.1 Dump Record Standardization

Debug dumps must contain metadata to describe the dump using header information, The format of the debug dump records can utilize the *Common Platform Error Record* (CPER) as described in Appendix N of UEFI specification.

At a high level a dump record consists of a record header followed by one or more fixed-length error record section descriptors. For each section descriptor, there is an associated variable-length section that contains either error data or informational data. An error record must contain at least one error record section.

The following diagram shows the general format of a dump record.

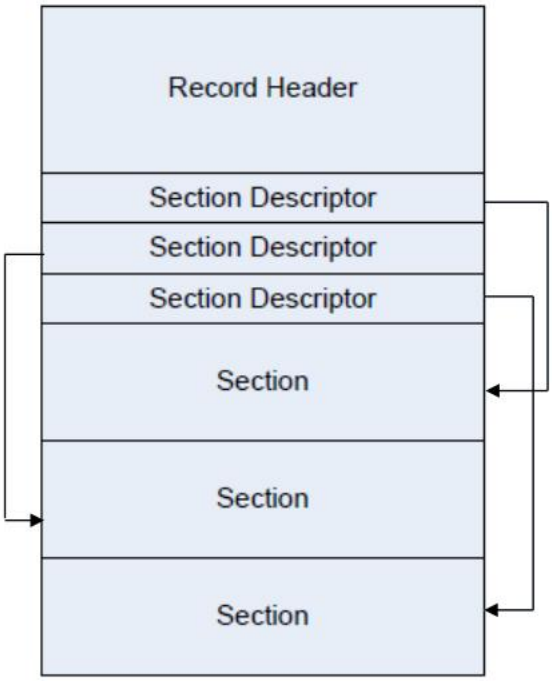

Figure 5: *Dump Record Format*

6.3 Debug dump tool analysis Requirements

This document assumes that debug dump records utilize a CPER format. It also assumes that silicon vendors provide tools that analyze the CPER errors logs and guide the Cloud Service Providers (CSP) to handle the error, e.g., do nothing, replace a part, invoke a RAS feature to mitigate the issue.

The requirements for vendor-provided analysis tools shall build on the OCP Fault Management Infrastructure Requirements Ver 1.0 which has been submitted by the OCP Hardware Fault Management Team.

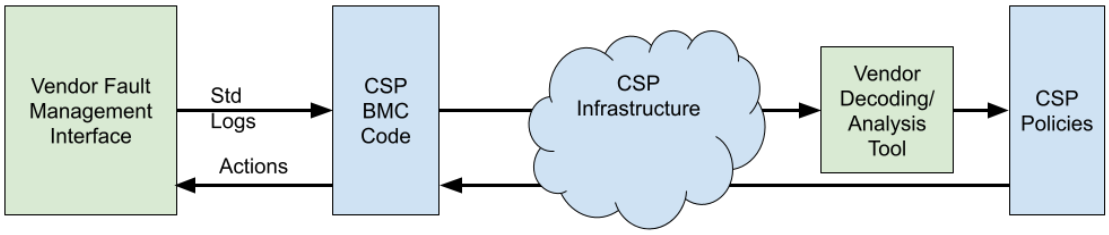

Figure 6: Overview of Dump Record Analysis

6.4 Analysis Framework

CSPs integrate and deploy vendor analysis tools into their infrastructure to interpret the CPER records generated. CPER records may differ across devices from different vendors, which require CSPs to properly route them to the corresponding analysis tool instance.

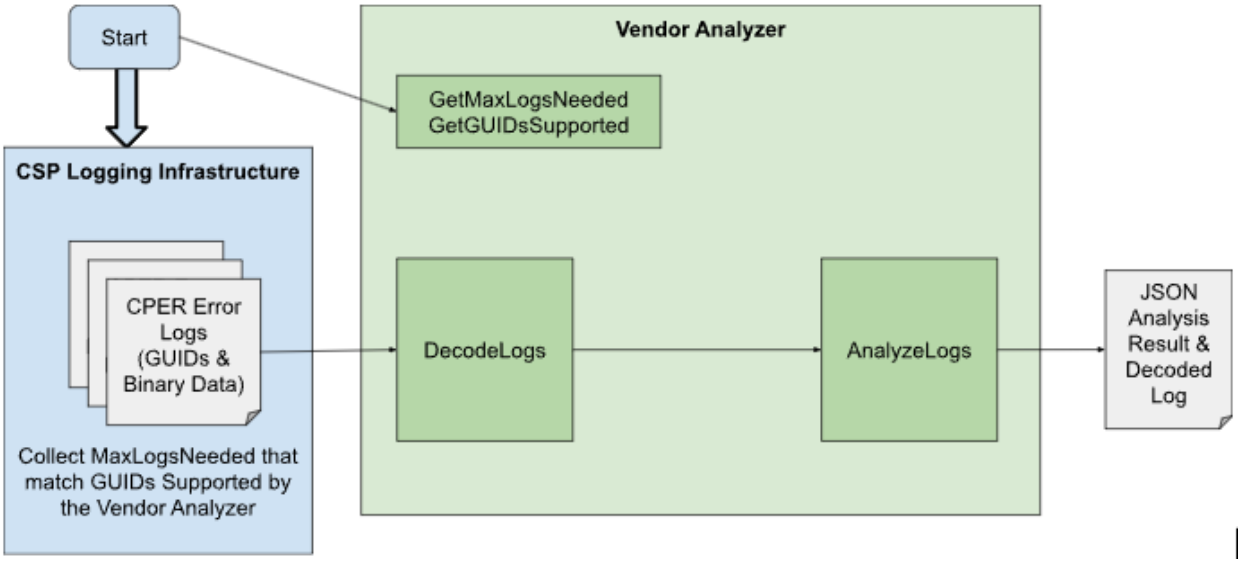

Figure 7: Dump Record Analysis Process

There are three categories of analysis tool requirements to ensure their interoperability with the CSP infrastructure:

1. Analyzer code requirements
2. Analyzer Input format requirements
3. Analyzer Output format requirements

6.4.1 Analyzer Code Requirements

1. Analysis tool shall be self-contained, with all dependencies included in the package delivered.
1. Analysis tool implementation shall be compatible with Cloud deployment practices such as packaging and running within a container.
2. The vendor analysis tool shall expose an API interface to enable the CSP infrastructure to discover its capabilities:
    a. CPER Section types can decode by responding with a list of the CPER Section GUIDs supported.
    b. Whether time-series data is needed by the analysis tool to properly identify the cause of the error.
    c. Any additional parameters required to enable CSPs infrastructure configuration to automate feeding CPER records to the analyzer tool.

6.4.2 Input Format Requirements

The vendor analysis tool shall consume dump records in a CPER format, as previously stated in this document.

6.4.3 Output Format Requirements

1. Output should be in JSON format that is both machine-parsable and human-readable.
2. Output shall include a section containing decoded data from the CPER records, with register-level decoding as a minimum, and register field-level decoding as preferred.
    a. Register-level decoding example: `"aer_uncor_status" : "0x00001000"`
    b. Field-level decoding example:

```
"aer_uncor_status" : {
     "dlp_err" : "0",
     "surpdn_err" : "0",
     "poison_tlp_err" : "1",
       …
```

}

3. Output shall include a section containing error data analysis results, including:
    a. Metadata decoded from CPER record (e.g., CPER version, FW version, timestamp, device type identifier, etc.) and metadata from the analyzer tool (e.g. tool version, tool-related timestamps, etc.).
    b. A failure identifier that CSPs can refer to when talking to vendors.
    c. The identifier of the IP block that logged the error.
    d. Suggested actions to take (e.g., replace part, update FW, invoke RAS feature, contact vendor), along with pertinent information to complete the action (e.g. FRU to replace, FW version to update to, RAS feature ID to invoke).
    e. Urgency of suggested action (e.g., as soon as possible, as convenient, before the next reboot, etc.).
    f. Confidence level of the analysis result.
4. Fields in the output report intended for machine-readability must serve a single purpose and not require regex parsing.
5. Desirable: analysis tool should have the capability to identify design defects known by the vendor and report them accordingly.

To facilitate standardization of these requirements, this document proposes to follow the JSON schema defined by the OCP HW Fault Management sub-project.

7. RAS Feature Requirements

GPU & Accelerators-based systems require various RAS features to support the goals as described in chapter 5. Refer to Appendix A for a high-level summary of the RAS features. The following sections describe key RAS features, e.g., PCIe fault management and GPU fault management in more detail. The future release of this document will describe additional RAS feature requirements.

8. PCIe Fault Management and RCA Telemetry

PCIe faults should be handled appropriately so that system level service interruptions are minimized, up-time is maintained, and data corruption is prevented. Robust PCIe corrected and uncorrected error reporting is

required for preemptive maintenance action and RCA. PCIe Error handling is required to be compliant with PCI Express Specifications, including AER and DPC.
Although PCIe AER support is optional in the PCIe specification, for GPU & Accelerators-based system designs this should be mandatory. For example, DPC support (containment and recovery) is required at the CPU Root port, PCIe switches, downstream ports, and endpoints such as GPU, NIC, and NVMe drives. Current systems enable DPC at the system level for all Root Ports. We require support for enabling DPC individually in each Root Port and downstream port (e.g. on switches) including support in HW, FW and OS.

Other critical requirements are to support system performance telemetry related PCIe errors such as Error storms and PCIe link downgrade. PCIe Link degrade errors should be supported with MSI signaling as per the PCIe specification.

In addition, we have a system level requirement, that the CPU root port has RP_PIO support to handle PCIe errors as per the PCIe specification.

Additional pain-points and detailed requirements will be further clarified as part of OCP HW Fault Management workgroups and in future releases.

9. GPU Fault Management software requirements

Our goal is to improve the way software handles hardware errors in order to minimize the impact of the hardware errors. This will improve up time and reduce checkpointing overhead. This section will define features and interfaces for RAS actions that minimize the impact of hardware errors, including:

- Poison handling flow Improvements (Drivers)
    a. For the containment of corrupted data.
    b. To enable recovery from poison, potentially failing an operation, but not crashing the device/system.
- Runtime HBM Post package repair.
- HBM page off lining.
- Ability to offline parts of the GPU, e.g. swapping out a bad core.
- Surprise Hot-plug support to enable end-to-end DPC flow for containment and recovery from PCIe UCE such as ‘Rx Poison TLP.
- Improve RAS actions based on fleet data.

Detailed requirements will be further clarified as part of future releases.

10. Compliance tools

The OCP Test & Validation Initiative is a collaboration between datacenter hyperscalers having the goal of standardizing aspects of the hardware validation/diagnosis space, along with providing necessary tooling to enable both diagnostic developers and executors to leverage these interfaces. Based on this, a tool called CTAM (Compliance Tool for Accelerator Management) has been developed to work with this and other related specifications, and provides compliance testing for Accelerator Management in cloud data centers.

The tool will check compliance of accelerators to as many parts of this RAS specification as is possible in an automated manner. The idea being that hyperscalers and suppliers of accelerators can measure the level of compliance of their hardware with their stacks. The compliance tool will focus on using Redfish to validate the following areas of this specification. The scope will increase in future specifications.

1. Discovery of available debug dump mechanisms (see section 6). At least one debug dump mechanism should be discoverable.
2. For the discovered debug dump mechanisms, perform the debug dump actions (see section 6).
3. Discovery of error injection mechanisms (see section 4). At least one error injection mechanism should be discoverable.
4. For the discovered error injection mechanisms, perform the injection followed by validation of action and response (see section 4)

11. References

- PCI Express Base Specification, Revision 4.0, September 27, 2017.
- Unified Extensible Firmware Interface Specification, version 2.9,
- OCP GPU & Accelerator Management
- FW Update Specification.
- CPER record reporting through Redfish are referenced here: https://www.dmtf.org/sites/default/files/Redfish_LogEntry_for_CPER_WIP.pdf.
- OCP Fault Management Infrastructure Requirements Ver 1.0

- OCP HW Fault Management sub-project

12. OCP Tenets

Openness

This specification has been developed using 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 on quality and TTM for GPU Systems by Hyperscalers. Furthermore, it is expected to be applicable to Enterprise deployments as well.

Scale

This 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.

Appendices

Appendix A – RAS Feature Requirements

The table below contains summaries of key RAS features requirements. These are examples only and not intended to be normative.

SN	Component	RAS Feature	Scope	Comment
1	Memory	ECC (SECDED)	TBD	
2		ECC (Block Correction Codes)	TBD	
3		Row-remapping (aka PPR) – boot-time	TBD	
4		Row-remapping (aka PPR) - run-time	TBD	

5		Patrol Scrubbing	TBD	
6		Command Address Parity Protection	TBD	
7		Link Error Protection	TBD	
8		Data Poisoning for containment	TBD	
9		Corrected Error Reporting	Required	
10		Uncorrected Error Reporting	Required	
11		Error Injection (in-band)	TBD	
12		Error Injection (out-of-band)	Required	
13	Cache	ECC (SECDED or better)	TBD	
14		Data Poisoning for containment	TBD	
15		Corrected Error Reporting	Required	
16		Uncorrected Error Reporting	Required	
17		Error Injection (in-band)	Required	
18		Error Injection (out-of-band)	Required	
19	GPU	All 1's data exception handling to prevent data corruption	Required	
20		Rx Poison TLP and Egress Poison TLP handling to prevent data corruption	Required	
21		Ability to hot-reset core(s) without affecting rest of the platform	TBD	
22		Error Injection (in-band)	Required	
23		Error Injection (out-of-band)	Required	
24	PCIe I/O	PCIe specification-based error handling including "Advanced Error Reporting (AER)"	TBD	
25		End to end CRC (ECRC) support	TBD	
26		Rx Poison TLP and Egress Poison TLP error reporting	TBD	
27		Error Injection (in-band)	TBD	
28		Error Injection (out-of-band)	Required	
29	Interconnect and Switches	Vendor specific link error handling (including reporting and recovery)	TBD	
30		Error Injection (in-band)	TBD	
31		Error Injection (out-of-band)	TBD	
32	Re-timers	Vendor specific link error handling (including reporting and recovery)	TBD	

33		Error Injection (in-band)	TBD	
34		Error Injection (out-of-band)	TBD	
35	Kernel (AER/DPC/HP Handler)	Kernel AER/DPC/HP drivers supporting PCIe FW specification requirements including EDR (Error Detect and Recovery)	TBD	
36	GPU Driver (Error Handler)	Surprise Hot-plug support to enable end-to-end DPC flow for containment and recovery from PCIe UCE such as 'Rx Poison TLP, 'CTO', UR....	TBD	
37		Ability to handle PCIe switch virtual port in case of surprise link down	TBD	
38		Error Reporting (in-band)	Required	
39		Error Reporting (out-of-band)	Required	
40	AMC	Error Collector for out-of-band monitoring using CPER format (telemetry)	TBD	
41		Crashdump collector in case of module detecting a fatal error or machine crash	TBD	
42		Support Redfish interface towards hyperscaler infrastructure (Host BMC pass-through)	Required	
43		Support Redfish interface towards hyperscaler BMC	TBD	
44	In-band Management Agent	Error Collector for in-band monitoring using CPER format (telemetry)	TBD	
45		Ability to collect existing AMC stored Crashdump	Required	
46		Support Redfish interface towards hyperscaler infrastructure	Required	
47	Hyperscaler BMC	Support Redfish interface towards hyperscaler BMC	Required	
48	System SW	Check-pointing optimization to minimize risk of data loss and improve the uptime.	TBD	

Appendix B - Implementation Considerations

Using CPER as hardware error format

The following table shows examples of how CPER format can be used to report hardware errors in this framework.

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 and other errors	Section N.2.3 in UEFI spec Ver 2.9	9.2.10 Non-standard Section Body

In addition to standard CPER sections defined in the UEFI spec, vendors can also use non-standard sections to provide additional information to enhance the Error records.

The following provides an example of PCIe error data reported in the CPER PCIe error section:

```
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
```

Reporting CPER records via Redfish

The following provides a LogEntry object example that reports a small CPER record as inline diagnostic data:

```
{
"@odata.type": "#LogEntry.v1_14_0.LogEntry",
"Id": "3",
"Name": "CPER Log Entry with small diagnostic 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",
}
```

Large CPER files shall use the "AdditionalDataURI" property as proposed in

https://www.dmtf.org/sites/default/files/Redfish_LogEntry_for_CPER_WIP.pdf

Memory Error Injection Schema

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!---->
<!--################################################################################-->
<!--# Redfish OEM Schema:  OCPMemory  v0.7.0                                       -->
<!--################################################################################-->
<!---->
<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="InjectCorrectableECCError" IsBound="true">
<Annotation Term="OData.Description" String="Injects a correctable ECC error
to a specific memory address in the memory device."/>
<Annotation Term="OData.LongDescription" String="This action shall inject a
correctable ECC error to a specific 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 physical address in
which to perform a correctable ECC error injection as a hex-encoded string."/>
<Annotation Term="OData.LongDescription" String="This parameter shall contain
the device physical address in which to perform a correctable ECC error
injection as a hex-encoded string."/>
<Annotation Term="Validation.Pattern" String="^0x[0-9a-fA-F]+$"/>
</Parameter>
</Action>
<Action Name="InjectUncorrectableECCError" IsBound="true">
<Annotation Term="OData.Description" String="Injects an uncorrectable ECC
error to a specific memory address in the memory device."/>
<Annotation Term="OData.LongDescription" String="This action shall inject an
uncorrectable ECC error to a specific 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 physical address in
which to perform an uncorrectable ECC error injection as a hex-encoded
string."/>
```

```
<Annotation Term="OData.LongDescription" String="This parameter shall contain
the device physical address in which to perform an uncorrectable ECC 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 InjectCorrectableECCError and InjectUncorrectableECCError actions."/>
</Schema>
</edmx:DataServices>
</edmx:Edmx>
```

PCI-e Device Error Injection Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!---->
<!--##################################################################-->
<!--# Redfish OEM Schema:  OCPPCIeDevice v0.7.0                      -->
<!--##################################################################-->
<!---->
<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>
```

Appendix C - 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. So Modern systems 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 the 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 - 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 BIOS or UEFI firmware.
OS	Operating System - that runs 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, the software might see a series of corrected errors, all from the same row address. The software could decide that the 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.