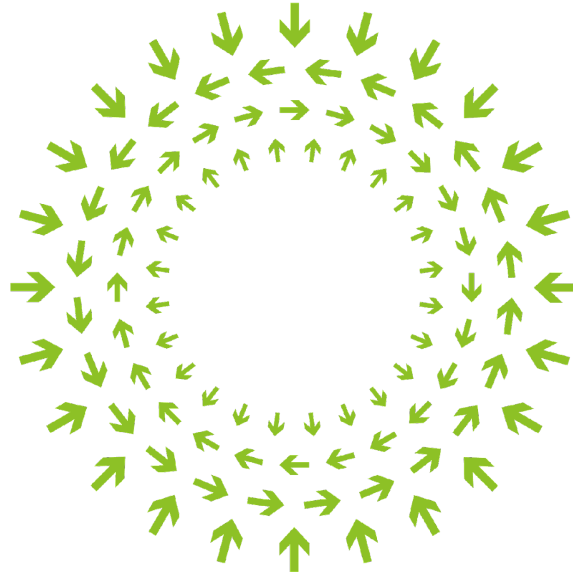




OPEN

Compute Project

RAS API Revision 0.8

Version 0.0

February 2023

Authors:

Intel Corporation (Contact Point: Antonio Hasbun)

Google LLC (Contact Point: Drew Walton)

Table of Contents

Contents

1. License	5
2. Compliance with OCP Tenets	7
2.1 Openness	7
2.2 Efficiency	7
2.3 Impact	7
2.4 Sustainability	8
4. Version Table	9
5. Scope	10
6. Overview	11
6.1 Problem Statement	11
6.2 Expected benefits	11
7. Requirements	12
7.1 General architecture	13
7.1.1 CXL as a base	14
7.1.2 Mailbox	14
7.1.3 Error records	16
7.2 Use cases	17
7.2.1 OS management	17
7.2.2 OOB management (BMC/SNIC/IPU)	18
8. RAS API platform integration	18
8.1 Agents and their scope	18
8.2 The challenge of mailbox ownership	19
8.2.1 Protecting the ownership of the mailbox	20
8.3 OS management integration	21
8.4 OOB management integration	22
8.4.1 SNIC/IPU integration	23
8.5 Device specific drivers	24
9. Opcodes	26
9.1 Features	27
9.2 Events	29
9.3 Timestamp	34
9.4 Trigger Action	35
9.5 Information and Status	35
9.6 New OPCODES	37
10. Features	39
10.1 Feature use modes	40
10.2 List of RAS features	41
10.2.1 Memory features	42

10.2.2	<i>Link features</i>	44
10.2.3	<i>Core features</i>	45
10.2.4	<i>Error injection</i>	45
11.	Error logs	45
11.1	Event record formats	46
11.1.1	<i>DRAM Memory error record</i>	46
11.1.2	<i>CPER error record</i>	46
12.	References (recommended)	48
13.	Appendix A - Checklist for IC approval of this Specification (to be completed by contributor(s) of this Spec)	49
15.	Appendix B-__ <supplier name> - OCP Supplier Information and Hardware Product Recognition Checklist	50
17.	Appendix C - ACPI table example for in band CPU RAS agent discovery	52

Figures

Figure 1 RAS API general architecture	14
Figure 2 Register mailbox from CXL spec	15
Figure 3 MCTP mailbox from the CXL spec	15
Figure 4 Software integration RAS API	19
Figure 5 OS RAS management integration	21
Figure 6 IB RAS API integration with legacy OS	22
Figure 7 OOB BMC RAS API integration	23
Figure 8 IMC RAS API and system integration	24
Figure 9 SNIC/IPU RAS API integration	24
Figure 10 GPU integration using device specific driver	25
Figure 11 GPU proposed RAS API integration	26
Figure 12 Feature discovery	28
Figure 13 Auto Memory Sparing - Autonomous mode	40
Figure 14 Auto Memory Sparing - Assisted mode	41

Tables

Table 1 Initial permissions for mailboxes ownership	20
Table 2 Opcode leveraged from CXL	27
Table 3 Common feature attributes	29
Table 4 Get Event Interrupt Policy Output payload	31
Table 5 Set Event Interrupt Policy Input payload	33
Table 6 Get MCTP Event Interrupt Policy Input payload	34
Table 7 Identify Output Payload	36
Table 8 Input payload for Request Feature Ownership	38
Table 9 Output of Request Feature ownership	38
Table 10 Input payload for Release Feature Ownership	39
Table 11 Output of Release Feature Ownership	39
Table 12 Memory RAS features	42
Table 13 Memory RAS maintenance capabilities	43
Table 14 CPER Event Record	47
Table 15 ACPI In bound discovery table	52
Table 16 RAS API Configuration Structure	53

1. License

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

Intel Corporation
Google

Usage of this Specification is governed by the terms and conditions set forth in **Open Web Foundation Modified Final Specification Agreement (“OWFa 1.0”) (“Specification License”)**.

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

Notes:

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

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

NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2. Compliance with OCP Tenets

2.1 Openness

The publication of RAS API helps standardize the interfaces for RAS features on the platform, while providing a wide variety of knobs so hardware vendors can still innovate and differentiate with RAS features. The main goal of this spec is to facilitate the adoption of RAS features to end customers; making it open while still providing the flexibility of knobs it is expected to allow OEMs and CSPs to develop their management systems around this spec.

2.2 Efficiency

Current RAS implementation are proprietary and a great majority of them require firmware expertise to get deployed. The aim of simplification and hardware abstraction that are achieved through RAS API permits smaller teams to experiment and deploy RAS features easily and fast. This should provide a huge improvement in efficiency for the RAS teams involved.

Also, the standardization of error logs from any type of IP through a common API allows for more energy to be used in the analysis and use of the data; rather than in how to extract it; create yet one more efficiency for end customers.

2.3 Impact

Even though the RAS API was developed by Intel and Google; a great amount of the needs and problem statements come from the hardware fault management workstream at OCP. The specification will provide a huge advantage to end customers in their validation and deployment cycles and the ease of use for RAS features.

2.4 Sustainability

A paper by Google [1] analyzed and projected the number of resources wasted due to job failure and relaunched. That study indicated that 12%-20% of compute resources could be saved from current calculations if the right algorithms for predicting job failures were used. RAS API provides the standardization needed to further those analysis and apply them more broadly to any component in the platform. The savings on compute power will reflect positively not only in energy resources, but on TCO for datacenters.

3. Version Table

OCP uses a Revision-Version nomenclature where Revision is the major version and Version is the minor version.

Date	Version #	Author	Description
February 2023	0.8	Intel Corporation Google LLC	Initial specification released to OCP.

4. Scope

This specification covers the API for RAS features. This includes several services:

- Error collection and logging

- RAS policy enforcement across fleet

- RAS action triggering and flow execution (see use case: autonomous)

- RAS configuration handling

- Special RAS services

The specification shall cover all different type of devices in the platform and will not be limited to the CPU.

The specification shall **not** cover synchronous failure. Synchronous failures are defined here as the ones that require immediate handling and signaling and that must interrupt the normal core execution, for example, core poison consumption. The timing problem that arises from stopping core execution when such errors are encountered are beyond the scope of this specification and it will be assume that those features are handled by traditional means.

Furthermore, this specification is concerned with the RAS API definition. It will provide example of platform integration and several methods on how the OS, or the BMC can handle the interfaces; but it is not trying to specify how the platforms get connected nor how the software implements the connections beyond the driver.

5. Overview

5.1 Problem Statement

Over the years server RAS feature complexity has exponentially increased in configuration as well as run-time handling. Managing RAS has become a very complicated subject that requires subject experts and a steep learning curve from both hardware providers as well as customers. The two main areas of increased complexity are the feature configuration and changes with each generation. Configuration includes several possible signaling methods or fine-tuning parameters, as well as cross matrix of interaction among RAS features or with other server features like security. The changes for each generation are not due the lack of architectural structure, but more because of the increments in the number of IPs that need to be protected by RAS that forces the hardware manufacturers to find creative ways of using those architectures. Specifically, hardware vendors must make some changes to the Machine Check Bank allocations with each generation; and even though the banks themselves are architectural in nature, their constant change forces the customers to modify their RAS management software in each generation.

The RAS API strives to create a standard software interface that will provide a discoverable and extendable interface and method for customers to manage their RAS features at fleet scale. The RAS API will also provide a way to minimize the runtime RAS interactions that are needed. This will be achieved by providing a mailbox mechanism that does not require SMM handling and can manage most of the RAS flows.

5.2 Expected benefits

The API will enable at least the following benefits:

Reduce the need for SMI runtime flows to a minimum

In the past the Firmware First RAS implementation had some of the RAS flows in firmware allowing the OEMs to better tweak the RAS management to their specific needs. But the System Management Interrupts (SMI) that are the basis for that methodology today can potentially create undesired tradeoffs on performance and security. Therefore, we are designing the API with paths of communication between the hardware and the management software that do not make use of this SMM infrastructure.

Lower the RAS investment in software across platform generations

The API shall be architectural enough that no changes to the software infrastructure be required from one platform to the next one. The ideal case would be that the API is part of a standard across different companies, such that changing it would not be as dynamic as the change from one product to the next for any given company.

Leverage the ecosystem standardization

The API should leverage part of the infrastructure that is already well adopted in the server community to avoid the ramp into a particular technology and make sure that several companies will align to that single standard to manage RAS features.

Improve error data collection

The API should implement the best possible error collection standard and include as many details as possible. It should also allow for future expansions of error logs and more details to include segment specific error logs whenever needed. This will enable ML and AI RAS strategies that rely on error data collection.

Provide observability of RAS events to multiple agents

The API shall provide ability for observability to multiple agents other than the agent managing RAS. This is an important feature for CSPs who may want to have additional observability, however unable to do so due to common resources shared between the RAS agents cause race conditions.

Extensible to all sub systems in the platform

The API should be defined with a broad enough scope to encompass any and every sub system in the platform in their different possible RAS features. The datacenter is moving into a disaggregated model, and RAS management should be able to be homogenous across all its components.

6. Requirements

The basic requirements for this API are listed here:

Enumeration of features: (The API shall allow discovery and enumeration of features)

Since the API is generic, the feature must be discoverable, and the list of such features must be capable of expansion for future technologies

Architecturally defined: (The API shall be architecturally defined)

The software investment in the RAS fleet management is meant to be reduced significantly by having the same architectural spec every generation. This will also allow for a higher level of abstraction for the software; helping that insulation and reducing any possible changes to the Ras infrastructure only to cover new features or domains that are introduced.

Furthermore, the architecture must be open source so all customers can develop their own tooling and keep an industry standard that can guarantee the compatibility with future generation platforms.

Allow for autonomous and assisted modes (The API shall allow for autonomous and assisted modes)

The API should support the capability of specifying autonomous or assisted modes for each feature. The features should not necessarily implement both options, but the API should provide a method for the implementations to expose the full capabilities when enabled.

Extensible for RAS features across all the portfolio in the platform

This API should consolidate as well as abstract the RAS functionality to include all the possible components of the platform and still be flexible enough to accommodate for any future devices or RAS features in the future.

6.1 General architecture

The API is a software specification that follows the general architecture of Figure 1.

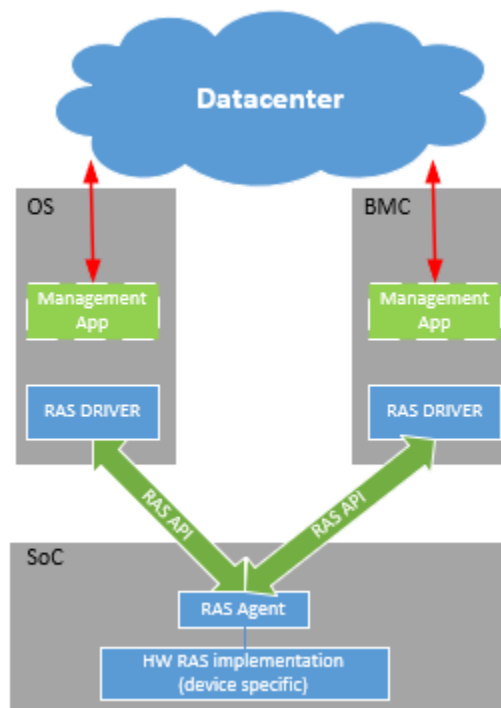


Figure 1 RAS API general architecture

In general, the device providing the RAS services should have at least two mailboxes that will allow for the control and management of the RAS features. The entity that exposes and controls the two mailboxes is what we would refer to as the **SoC RAS agent**. This specification keeps the API agnostic to the hardware implementation. The agent can be implemented as a micro controller with firmware, as a set of micro controllers or as a part of firmware inside an existing microcontroller. However, the specific implementation is left for the hardware design teams to decide, while the API will strive to document the software interfaces and interactions.

6.1.1 CXL as a base

CXL is an industry standard that provides several advantages to jump start the adoption of the RAS API. It is widely adopted and contains a very generic mailbox mechanism that can be discovered as a PCIe device easily. It also has a methodology to extend most of its functionalities providing backwards compatibility through versioning.

The leverage of CXL will start using the mailbox in order to facilitate the definition and discovery of the command opcodes between the management software and the device agent. The section in the CXL 2.0 spec for the definition of the mailbox registers is the chapter 8.2.9.4. From this section we leverage the mailbox definition as well as all the mechanics used to issue the commands and their responses.

The next section that we leverage is the feature definition. This functionality allows us to enumerate features (RAS features in our RAS API case). The section in CXL 2.0 spec is the 8.2.10.5. In this section the opcodes to "Get supported features" and "Get feature" are defined. These opcodes allow us to enumerate the RAS features that the device is capable to support and their attributes. We are also leveraging the "Set feature" opcode to configure the RAS features. These attributes can be read only to expose the capabilities of the device or writeable to configure the feature.

The observability of errors is going to leverage the event logs detailed in section 8.2.10.1 of the CXL 2.0 spec. The opcodes that are used for this are: "Get Event Record" "Clear Event Record" and the event interrupt handling opcodes: "Get Event Interrupt Policy" and "Set Event Interrupt Policy".

A more thorough description of all the OPCODES that RAS API leverages from CXL are described in chapter 5.

6.1.2 Mailbox

There are two types of mailboxes that will be leveraged from the CXL spec. The first one is the register's mailbox that mostly will use MMIO. The other one is the MCTP based mailbox that will use the CCI interface.

The register-based mailbox is described in Section 8.2.9.4 of the CXL spec, and it depicts the registers that are needed to communicate between the devices. Figure 2 shows as an example the representation of the registers that make up this mailbox. Notice that some registers are read-only from the host perspective.

Figure 8-13. Mailbox Registers

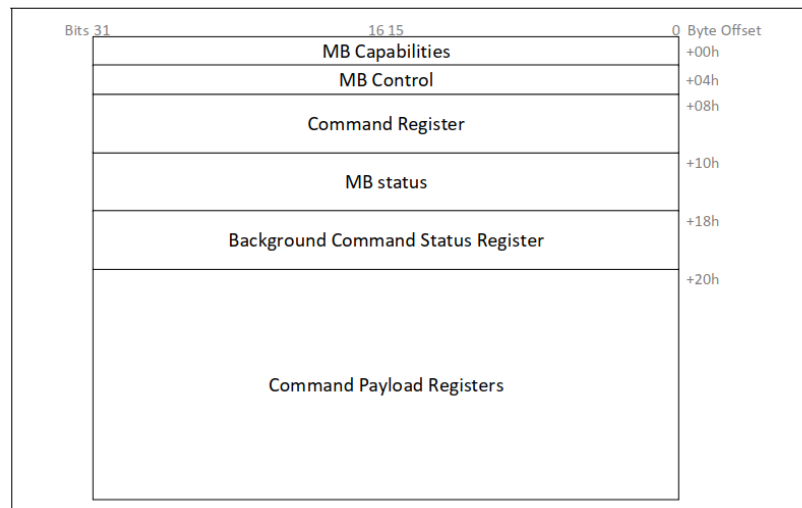


Figure 2 Register mailbox from CXL spec

The MCTP mailbox uses the CCI interface described in Section 7.6.3 in the CXL spec. In Figure 3 there is a snapshot for the MCTP CCI interface as an example.

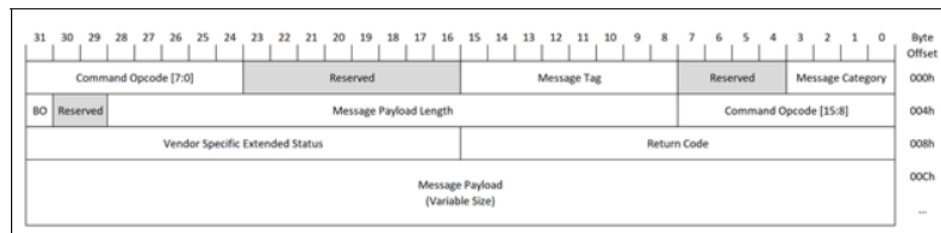


Figure 3 MCTP mailbox from the CXL spec

The mailboxes should implement the OPCODES as described in the Chapter 5.

The device must implement at least one of each type of mailbox, but in some cases the support for two IB mailboxes might be desired. When having multiple mailboxes there is an arbitration problem that needs to be addressed; in this spec in there are new opcodes being proposed to address that issue. Please see chapter 4.2 for details on how to manage the arbitration.

6.1.3 Error records

We are leveraging the CXL event record definition that is described here for the RAS API. The event records are classified by their severity and that gives the 4 different error queues present in the devices. Even though the CXL spec doesn't provide a guidance for each severity, this RAS specification will provide the following guidance:

Informational Event log: device detects a condition and can either correct the condition or recovery can be deferred or is not needed. A response is not typically required. The conditions being signal in this queue should not expect the queue to be configured to create interrupts to the host; so they should mostly be internal devices events with little to no impact on the platform.

Warning Event Log: device detects a condition and can either correct the condition or recovery can be deferred or is not needed. A response is typically not required or can be delayed. It is expected that Correctable Errors (CE) or AER severity 0 (ERR_CORR) be signaled in this queue. The events themselves do not prompt to action; but predictive algorithms can derive from these records to take some platform actions.

Failure Event Log: detects an error and is unable to correct or recover from it; certain transactions or data on the device can be lost, but the device is otherwise functional. A host response is typically required. This queue should include errors like Software Repairable Action Required (SRAR) or AER severity 1 errors. This queue is normally configured with some interrupt mechanism.

Fatal Event Log: device has become unreliable, fatal errors may result in the device going viral. A response is typically required. Errors in this queue include fatal uncorrectable errors (Fatal DUE with PCC=1) or AER severity 2 (ERR_FATAL).

Even though there are a fix amount of error queues determine by their severity, the error records within those logs are defined by a UUID. Specific UUID can be used to describe different errors and the standardization of the UUIDs error record format permits for a standard driver to process the error messages.

All logs shall return event records to the host in the temporal order the device detected the events in. So, the events happening earliest are returned first to the host. During this process there are **NO overwrite rules** as they were in the past RAS management systems. The reason for this is that the queues of different severities are separated so they won't conflict between themselves.

Even though the event records cannot be overflown a detailed count of them is expected to be kept by the agent. The Get error records command will include the flag for overflow with additional field for: count for overflow events, timestamp of first overflow event and timestamp for last overflow event. Together these three pieces of information should suffice to determine the sampling of error records that happens during an error storm.

The event logs can be configured to have different types of interrupts. It can have:

- No interrupt: this implies polling for these records from the host
- MSI/MSI-X
- FW interrupt (EFN VDM)
- MCTP message

This provides the maximum flexibility for the event record configuration on any of its external agent connections. Further details on how to configure these settings is presented in the section RAS API platform integration in chapter 4.

6.2 Use cases

To have the RAS API be a successful management method for RAS features, it needs to adapt to the current RAS management use case that exist; and provide its value under each of those scenarios.

6.2.1 OS management

For the OS first approach is when the datacenter management systems are connected through the OS. In this approach the OS is considered secure and therefore can run management software. It will contain or connect to the datacenter wide management systems that will have the policies and algorithms that need to be implemented on the node. The main interrupt method for error logs is going to be the MSI interrupts; and the mailbox that is being used is the MMIO one, and not the MCTP one.

Most of the agents that connect to the hardware RAS API will run in the OS. The integration proposal is detailed further in chapter 4.3

OS first approach is when the datacenter management systems are managed through the OS. In this approach the OS is considered secure and therefore can run management software. It will contain or connect to the datacenter wide management systems that will have the policies and algorithms that need to be implemented on the node.

6.2.2 OOB management (BMC/SNIC/IPU)

There are several methods for managing the RAS functionalities of a platform from the (OOB) Out of Band. The main idea on this method is to utilize an external agent to the host to control the RAS and link to the datacenter management tools. This is commonly achieved using BMC.

In this case the mailbox that is used is the MCTP mailbox and the error records use the MCTP message as interrupts. The integration proposal is detailed further in chapter 4.4.

7. RAS API platform integration

The integration of a management system for RAS can take many shapes depending on the end customer design decisions. It can range from a service running on the same host that determines the RAS actions and logs the errors to a distributed cloud service that maintains the datacenter fleet in synch. In all the cases the RAS API should integrate and abstract the hardware details for the management software to be able to take a more holistic approach.

In order to show a reasonable path to implement an integration with the platform for each of the major use cases of RAS API; this chapter presents the pieces involved as well as a deeper level on how they are connected to the platform. There is no intention to enumerate or solve all the possible platform permutations, just to show a reasonable path to each use case.

7.1 Agents and their scope

In Figure 4 the general architecture of the system with the software infrastructure is presented. It shows the major agents that participate in the integration.

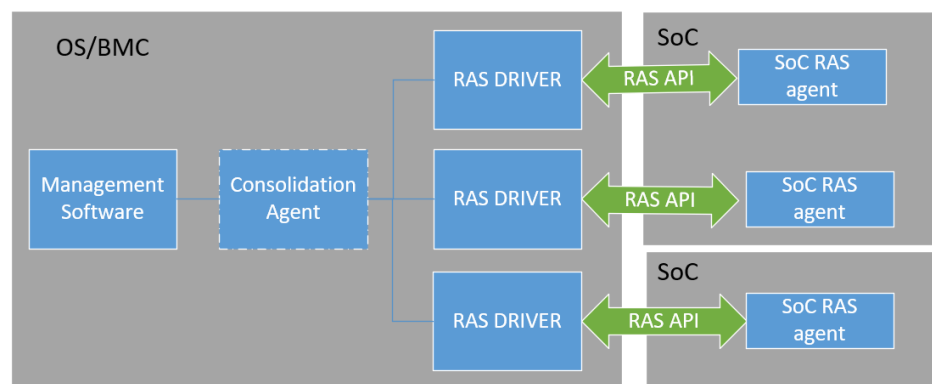


Figure 4 Software integration RAS API

It is important to note that in this figure that where the management software is being run makes no difference; it could be OS level, or in a bare metal instance it could be in a BMC.

There are a couple of new agents that are being proposed for the API integration. Let's look at the agents that exist in the systems and how they roles are impacted by RAS API:

SoC RAS agent: This is the function that controls the RAS mailbox from the devices' side. It is **hardware design specific**, but must adhere to the mailbox definition in this spec. it will expose several features depending on the hardware implementation of the RAS in the device.

RAS driver: This implements the low-level details of the connection interface. It spawns 1:1 with each mailbox that is discovered for this platform. Since the API is a std this piece of software is assumed to exist for each OS; and since it's not platform dependent it should be simple to maintain. The two main differences with today's implementation are that it is not platform specific and that it does need to spawn several times per platform.

Management software: This is the logic that analyzes the error and recommends the RAS actions. This includes things like Predictive failure analysis mechanism or even AI algorithms for fault prevention. This can be implemented in firmware or in the OS, or even split with parts also in the cloud (most noticeably the AI learning part). The management software works through the consolidation agent and finds specific data for the platform from it. This piece of software can be coded platform independent, but it is platform aware since many of the RAS algorithm depend on the technology and number of IPs in the platform.

Consolidation Agent: This piece of software can be optionally set as a standalone consolidation agent, or it can be integrated into the management software. Here is where the platform configuration gets exposed, and all the RAS mailboxes converge to provide a single interface for the management software.

The RAS driver is the most universal piece in the software stack. Since it is going to be part of an external specification it is assumed that a version of it will be available under different OSes and compiled to run on different processor architectures.

7.2 The challenge of mailbox ownership

One of the current challenges in the RAS management implementation is the need for a clear-cut ownership of RS fetures. There are several strategies for host management that can be used and each feature on the platform sometimes has a different one that it implements. The most common are driven out of OS, or Firmware or lately the BMC offloading. Since the RAS API enables mailboxes IB and OOB, there is a need to arbitrate which management agent will control the RAS features.

7.2.1 Protecting the ownership of the mailbox

The first thing that needs to be clarified is the assumption that spec makes about the platform design. There are basically two major assumptions on the system. First that the platform designer will guarantee that there is a single agent behind each mailbox. This means that the MMIO mailbox will have a single driver owning it, or that the MCTP mailbox will have a single source of commands for it. The second requirement is that firmware should set up the permissions for the mailboxes to allow or deny ownership of RAS features. This very initial step is very important in order to prevent any security compromise and in order to guarantee the system will behave within the limits set by the platform designer. An example of this permissions can be seen in the following table. The system in the example is meant to be managed by the OOB agent.

Feature/Event Queue	IB Mailbox	OOB Mailbox
Soft PPR	Deny	Allow
Hard PPR	Deny	Allow
Memory Sparing	Deny	Allow
Memory Mirror	Deny	Allow
Information Event Log	Allow	Allow
Warning Event Log	Allow	Allow
Failure Event Log	Allow	Allow
Fatal Event Log	Allow	Allow

Table 1 Initial permissions for mailboxes ownership

There are two types of resources that need to be protect for contention: the RAS features and the event logs. The RAS features require protection since two or more agents trying to trigger or change attributes on a single feature can cause it to malfunction. The event logs can be protected or not depending on the platform designer's preference, but the mechanism is provided in case a side channel vulnerability requires one of the mailboxes not to have access to the error records.

The mechanism described in this spec allows for control of individual RAS features and individual event logs. For the RAS features what gets controlled is the "Set feature" and "Perform maintenance" commands. The Get Features command provides discoverability and need not be controlled by this method. On the other hand, for the event logs what gets controlled is the "Get" and "Clear" events as well as the "Set Event Interrupt Policy". Only the mailbox that owns the feature or event log can issue the commands mentioned here. If other mailboxes attempt to use them, they will get "Unsupported" return message. Therefore, features that are not claimed cannot have their attributes set or maintenance commands issued.

For event logs it is possible that several mailboxes claim the log. Since each mailbox has its independent queue of error records, no contention should occur. But if the initial setup of

the agent has a policy that prevents one mailbox of obtaining the ownership of the error queue that mailbox won't be able to get or clear the error records.

7.3 OS management integration

The use case for OS integration includes the management through direct OS control.

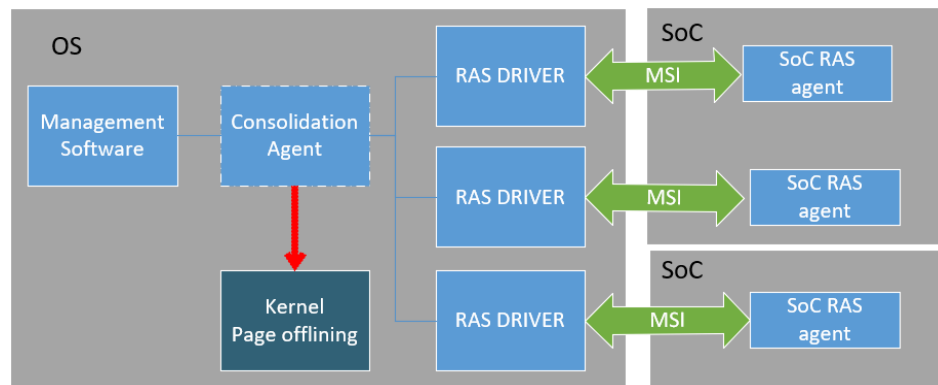


Figure 5 OS RAS management integration

The different SoC RAS agents connect to the corresponding RAS driver on the OS using the MMIO mailbox and using MSI as the interrupt mechanism. The error logs using the event logs should use the MSI or polling as required for the severity level.

There are two ways to discover the RAS:

Using a PCIe extension provides a mechanism for passing management messages between system software and a PCI Function using a mailbox interface in the Function's memory mapped I/O (MMIO) space. The software driver will use the Bus Device of the PCIe device to link the RAS interface with the functional device on the platform. If the RAS API uses the main function, then device driver will be in charge of loading the software driver for RAS API otherwise if it uses another function that will be register to RAS API.

Using the ACPI table – Processor RAS capability table. It provides not only the Generic Address Space (GAS) to the mailbox locations, but also the mapping of each RAS API to the corresponding APIC IDs so the software can match the functional unit to the RAS interface that is being provided. This is the preferred method for CPUs. In this case the OS will oversee loading the RAS API driver and it will have the associated devices (CPU or other Ips within the SoC) from the ACPI table. See example ACPI table in appendix A.

The software that interfaces with the RAS API should be the RAS API driver as designed for the specific OS that is running on the platform.

The management software from the datacenter or host can connect to the consolidation agent using Redfish. For memory features at least the kernel needs to connect to the consolidation agent to provide services like page off-lining. The connection between the kernel and the consolidation agent will be OS dependent; and probably for legacy OS a different architecture must be followed.

For integration without impacting the OS the architecture in Figure 6 is recommended. In this integration instead of communicating from the consolidation agent directly to the OS kernel for memory services; the _OSC is used to communicate with the BIOS and generate the GHEST entries that would feed the kernel the necessary information to perform page off-lining.

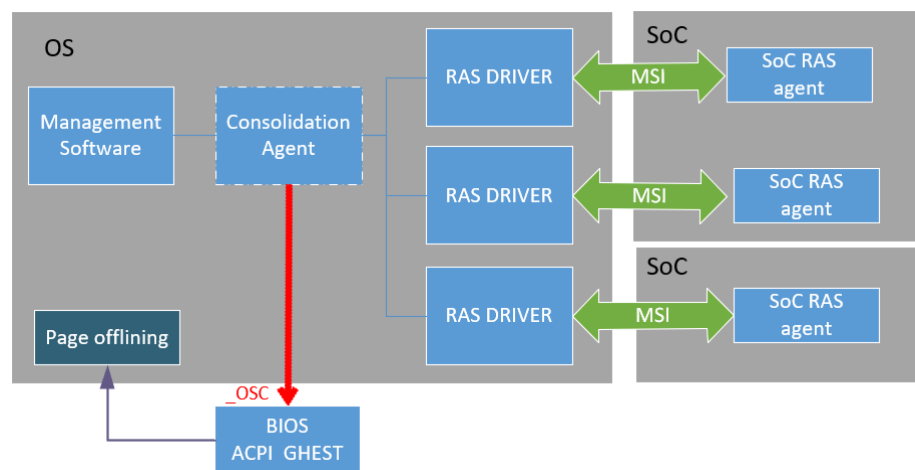


Figure 6 IB RAS API integration with legacy OS

7.4 OOB management integration

The OOB method utilizes the MCTP mailbox to communicate between the BMC and the SoC RAS agent. An MCTP message is used instead of normal interrupts for the error logs.

The discovery of the RAS API is done through normal MCTP methods. MCTP should use message type 7Fh that is IANA specific. The vendor type will be the OCP IANA number; in order to keep the specification of RAS API neutral to all providers. After the initial discovery, MCTP will also share a PLDM model that will convey the rest of the device configuration, especially the FRU information that might be needed in RAS actions.

Figure 7 shows the interconnections that are expected of the integration of the RAS API into the BMC manage platform. It is important to highlight that the connection from the BMC into the BIOS so that the BIOS can create the GHEST records for the OS where page off-lining or any other OS specific action must be taken.

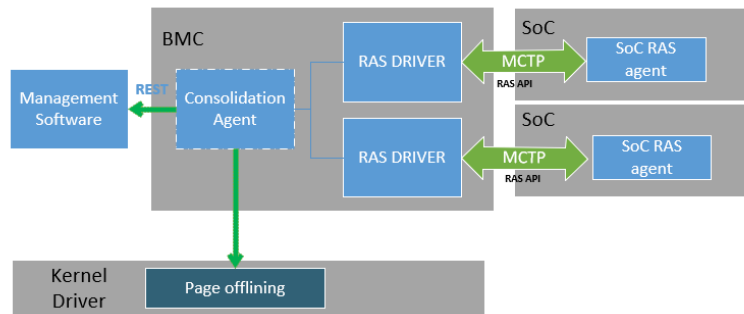


Figure 7 OOB BMC RAS API integration

7.4.1 SNIC/IPU integration

The RAS API integration for the Infrastructure Processing Unit (IPU) or a Smart NIC (SNIC) is a special case for the OOB RAS API integration. It is important to separate the offloading characteristics of the system as opposed to the RAS functionalities of the SmartNIC itself.

The first two types of RAS features that comes with the IPU are extensions of the RAS API into more SNIC specific RAS functions, but they are highlighted in this section to distinguish them from the new use case of offloading.

The first RAS extension is the RAS features of the compute complex itself that is different from the host CPU's core RAS functionality. In the following figure it can be seen how the cores that run the compute complex in the SNIC have different RAS features exposed through SoC specific mechanisms. For simplicity in the illustration, we call the microcontroller in charge of these activities within the SNIC the Integrated Management Complex (IMC).

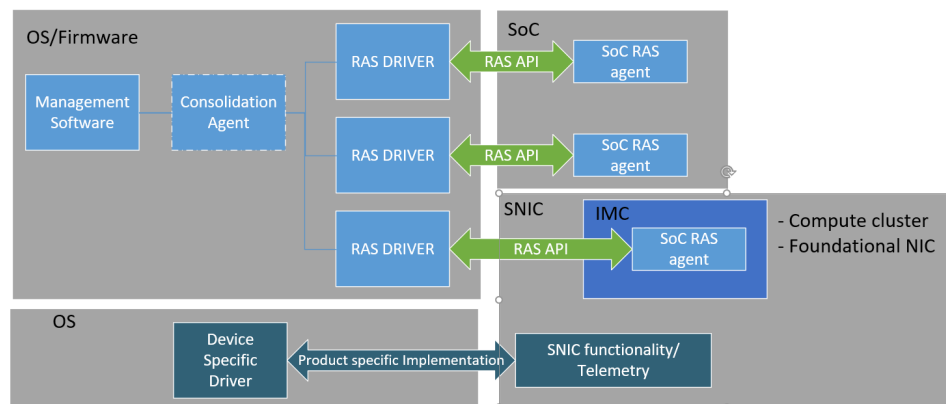


Figure 8 IMC RAS API and system integration

The memory controllers in the SNIC/IPU might not have the same RAS features as the host's CPU memory controller. The abstraction through RAS API will help the same agent manage this difference with little to no extra effort.

The second set of RAS features that are exposed through the IMC are the RAS features of the foundational NIC. Foundational NICs have error records and RAS actions like partition resets that are tracked and recorded through their IMC. It is important to note that the new features for the NIC have no architectural difference from the other RAS features, and as such are added to the RAS features in this spec as normal.

Next, we will analyze the offloading case, where the management software, or local connection to it is offloaded to a compute complex inside the SNIC. This case needs further path finding since the IPU offloading is not fully define and adapting RAS API would need a full definition of the IPU strategy.

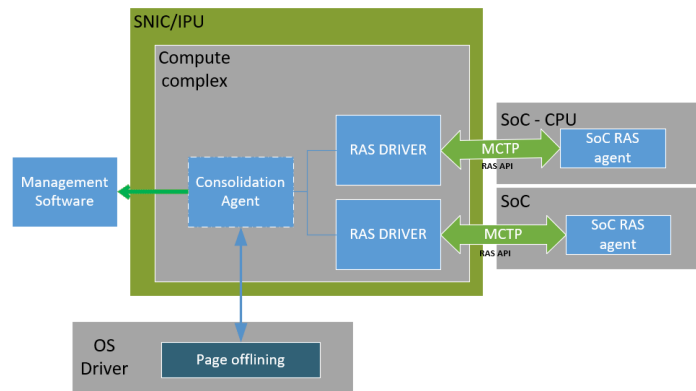


Figure 9 SNIC/IPU RAS API integration

The biggest difference with the BMC offload is the connection system. BMC is directly connected to most of the devices in the platform, while the IPU doesn't necessarily poses all the links.

7.5 Device specific drivers

One of the main challenges of creating a specification that can be adopted by all the devices on a platform comes from the traditional applications that today manage certain devices in a proprietary way. For example, there are GPU vendors that produce a software suite that includes a driver to manage their GPU. This suite of software includes the drivers for functionality, as well as software to manage the RAS of the device.

The current implementation of the RAS connection to management software is depicted in Figure 10. In some cases, the device specific driver provides connection from vendor tools

that then can be connected to the datacenter agents. In either case the only connection to the GPU hardware comes from device specific drivers.

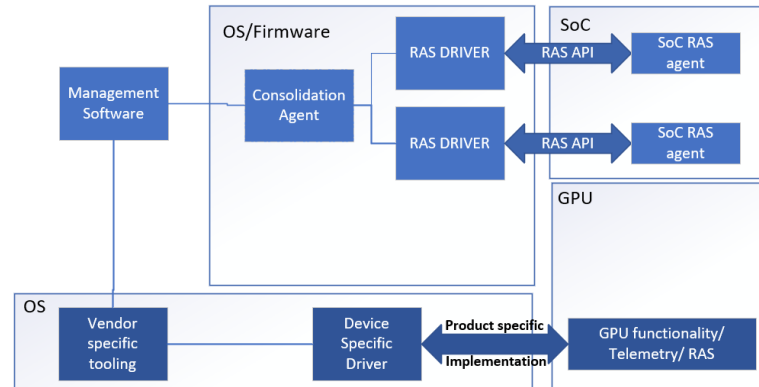


Figure 10 GPU integration using device specific driver

The proposed integration using the RAS API involves separating the RAS functionalities from the general GPU functionalities and providing an extended PCI functionality and an OOB mailbox to implement the RAS API.

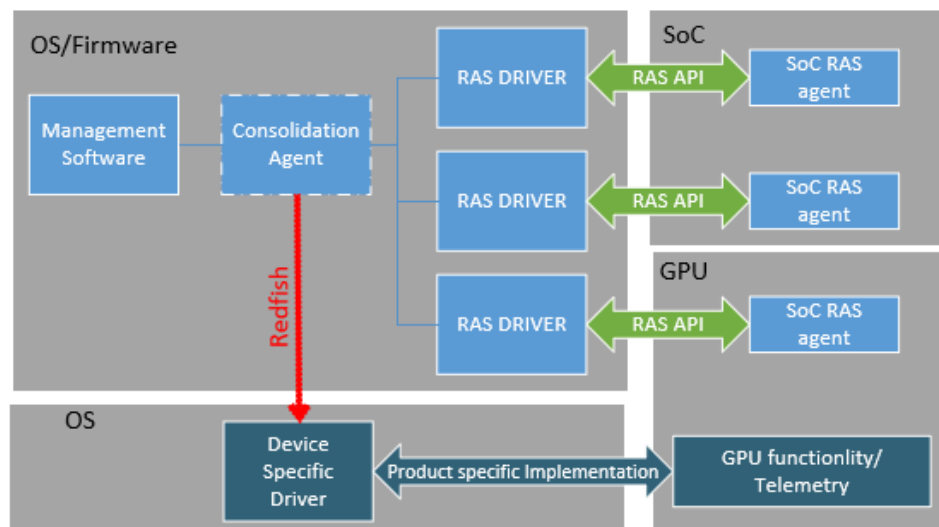


Figure 11 GPU proposed RAS API integration

This implementation provides a seamless integration of the device into the datacenter and it separates the RAS handling from the functional handling. Providing this OOB mailbox adds unique value for the GPU's and similar devices since it's the only way that device management can happen in bare metal instances within a data center.

8. Opcodes

To manage the RAS services the RAS API provides a set of OPCODEs that can be processed by the RAS agents to find, configure, and trigger the RAS services.

There are opcodes that are created uniquely for this RAS API as well as opcodes that where inherited from the CXL spec that we use as a baseline. This is the list of the opcodes that are being leveraged from the CXL specification. Some details of the opcode might have been tweak so further details are show in the following sections.

Group	OPCODE	Command	MMIO Mailbox	MCTP Mailbox
Information and Status	0001h	Identify (Section 8.2.10.10.1)	No	Yes
	0002h	Background Operation Status (Section 8.2.10.10.2)	No	Yes
	0003h	Get Response Message Limit (Section 8.2.10.10.3)	No	Yes
	0004h	Set Response Message Limit (Section 8.2.10.10.4)	No	Yes
	0005h	Request Abort Background Operation (Section 8.2.9.1.5)	Yes	Yes
	0015h	Request feature ownership	Yes	Yes
	0016h	Release feature ownership	Yes	Yes
Events	0100h	Get Event Records (Section 8.2.10.2)	Yes	Yes
	0101h	Clear Event Records (Section 8.2.10.2.1)	Yes	Yes
	0102h	Get Event Interrupt Policy (Section 8.2.10.2.2)	Yes	No
	0103h	Set Event Interrupt Policy (Section 8.2.10.2.3)	Yes	No
	0104h	Get MCTP Event Interrupt Policy (Section 8.2.10.2.4)	No	Yes
	0105h	Set MCTP Event Interrupt Policy (Section 8.2.10.2.5)	No	Yes
	0106h	Event Notification (Section 8.2.10.2.6)	No	Yes
Timestamp	0300h	Get Timestamp (Section 8.2.10.4.1)	Optional	Optional
	0301h	Set Timestamp (Section 8.2.10.4.2)	Optional	Optional

Features	0500h	Get Supported Features (Section 8.2.10.6.1)	Yes	Yes
	0501h	Get Feature (Section 8.2.10.6.2)	Yes	Yes
	0502h	Set Feature (Section 8.2.10.6.3)	Yes	Yes
Maintenance	0600h	Perform Maintenance (Section 8.2.10.7.1)	Yes	Yes

Table 2 Opcode leveraged from CXL

8.1 Features

The RAS API will leverage the CXL feature opcodes to provide discoverability of RAS features in the devices. The features are going to be describes in the spec and add to the CXL spec if they correspond to industry standard features or as vendor specific if they are specific to a hardware vendor.

The main opcodes to be used from the CXL spec definition are:

Get Supported Features (Opcode 0500h): This command allows to query the device for the list of all features supported. The features are identified by the UUID that corresponds in the spec, and their versions. No changes are required from this opcode from the CXL original definition.

Get Feature (Opcode 0501h): This command queries the attributes for a particular feature. Those attributes vary by feature and version and are specified in the spec. No changes are required from this opcode from the CXL original definition.

Set Feature (Opcode 0502h): This command configures the writeable attributes of a feature. The spec will show which attributes are writable depending on the version and the feature being queried. No changes are required from this opcode from the CXL original definition.

The following figure shows how the flow for querying the features of a device should be utilized by a host.

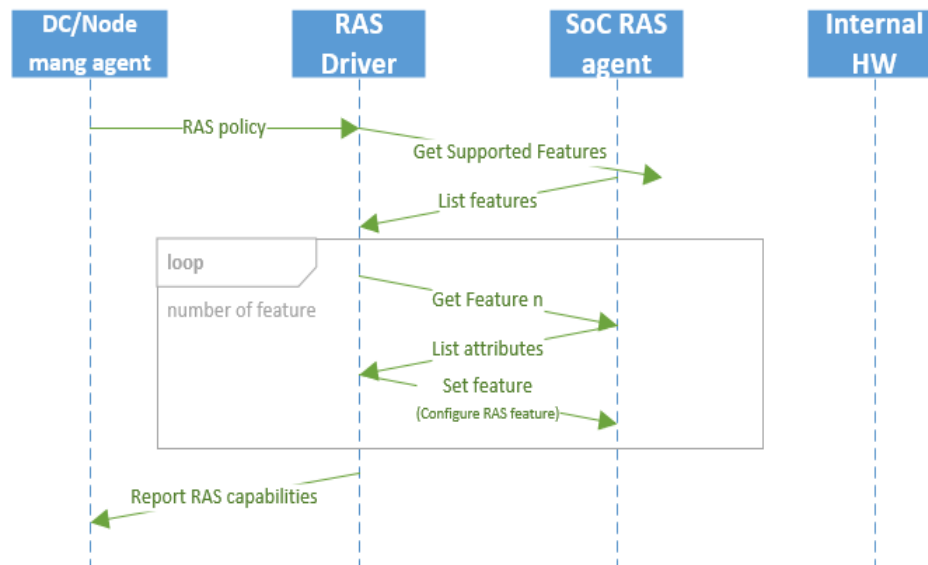


Figure 12 Feature discovery

It is important to note the way that this methodology allows for extending the features and their attributes in the future. The version of the features can be incremented when attributes are added to the feature, that way the feature keeps the backwards compatibility. In the case the host wants to address the features as a previous more limited version of it (perhaps because the driver is not updated to the lastest version of the feature) it just needs to indicate using the set feature a different version of the feature to use. When backward compatibility cannot be maintained a new UUID must be created to add a new feature to the list. This methodology provides an infinite capacity to expand the features and their attributes and future-proof the solution

All the features that are going to be defined for the RAS API will have the common maintenance attributes in CXL, which are specified on Table 8-86 for CXL spec 2.0; the are also reproduce here in Table 3.

Attribute	Description
RO	Maximum Maintenance Operation Latency
RO	Operation Capabilities - Bit [0] Device Initiated Capability
RW	Operation Mode - Bit [0] Device Initiated
RO	Maintenance Operation Class
RO	Maintenance Operation Subclass

Table 3 Common feature attributes

Note that the “Device Initiated Capability” is the use mode refer to as autonomous through this spec. The device advertises as a Read-Only (RO) attribute its capability to support the autonomous mode and the management software can set “Operation Mode” in the Read-Write (RW) attribute if they desire to use the autonomous mode. It is important to note that not all features will be capable of the none-autonomous mode or host-initiated mode. On those cases the “Device Initiate Capability” bit will be set as well as the “Operation Mode”; but when a Set feature command attempts to set the Operation Mode to 0 it will fail to change this attribute.

There is other limitation for the autonomous mode that are explained in Chapter 6.1.

It is also common to all the features for maintenance the maximum latency and the class/subclass classification.

8.2 Events

The RAS API will leverage the CXL feature opcodes to provide error logs in the form of event logs; this is discussed on more detail in Chapter 7.

The main opcodes to be used from the CXL spec definition are:

Get Event Records (Opcode 0100h): This is used in the MMIO mailbox to retrieve the event record on the device. It uses the flag “More Event Records” to highlight that there are more events that what fits in the payload of the mailbox. No changes are required from this opcode from the CXL original definition.

Clear Event Records (Opcode 0101h): This is the mechanism used to clear the event records that have been consumed. Its input payload has a “Number of Event Records Handles” and a “Event Record Handles” that list all the event records that will be removed. No changes are required from this opcode from the CXL original definition.

Get Event Interrupt Policy (Opcode 0102h): This command retrieves the current interrupt policy for device events. Each event log can have one of three different interrupt mechanism (no interrupts, MSI, EFN VDM).

The most important change is in order to enable interrupts to be generated not at the first event, but when the event queue has filled a percentage of the available queue. The options for the threshold can be controlled using the 2 bits available, from 0% which is the default in CXL to 75% in 25% increments.

We also need to set “Dynamic Capacity Event Log Interrupt Settings” to 00h since that event log is not implemented in RAS API.

Byte Offset	Length in Bytes	Description
00h	1	Informational Event Log Interrupt Settings: Specifies the settings for the interrupt when the information event log transitions from having no entries to having one or more entries. • Bits[1:0]: Interrupt Mode — 00b = No interrupts — 01b = MSI/MSI-X — 10b = FW Interrupt (EFN VDM) — 11b = Reserved • Bits[3:2]: Reserved • Bits[7:4]: FW Interrupt Message Number - Specifies the FW interrupt vector the device shall use to issue the firmware notification. Only valid if Interrupt Mode = FW Interrupt.
01h	1	Warning Event Log Interrupt Settings: Specifies the settings for the interrupt when the warning event log transitions from having no entries to having one or more entries. • Bits[1:0]: Interrupt Mode — 00b = No interrupts — 01b = MSI/MSI-X — 10b = FW Interrupt (EFN VDM) — 11b = Reserved • Bits[3:2]: Reserved • Bits[7:4]: FW Interrupt Message Number - Specifies the FW interrupt vector the device shall use to issue the firmware notification. Only valid if Interrupt Mode = FW Interrupt.
02h	1	Failure Event Log Interrupt Settings: Specifies the settings for the interrupt when the failure event log transitions from having no entries to having one or more entries. • Bits[1:0]: Interrupt Mode — 00b = No interrupts — 01b = MSI/MSI-X — 10b = FW Interrupt (EFN VDM) — 11b = Reserved • Bits[3:2]: Reserved • Bits[7:4]: FW Interrupt Message Number - Specifies the FW interrupt vector the device shall use to issue the firmware notification. Only valid if Interrupt Mode = FW Interrupt.
03h	1	Fatal Event Log Interrupt Settings: Specifies the settings for the interrupt when the fatal event log transitions from having no entries to having one or more entries. • Bits[1:0]: Interrupt Mode — 00b = No interrupts — 01b = MSI/MSI-X — 10b = FW Interrupt (EFN VDM) — 11b = Reserved • Bits[3:2]: Reserved • Bits[7:4]: FW Interrupt Message Number - Specifies the FW interrupt vector the device shall use to issue the firmware notification. Only valid if Interrupt Mode = FW Interrupt.
04h	1	Reserved
05h	1	Informational Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries — 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
06h	1	Warning Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries

		– 01b = transitions above the 25% of the queue capacity – 10b = transitions above the 50% of the queue capacity – 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
07h	1	Failure Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds – 00b = transitions from having no entries to having one or more entries – 01b = transitions above the 25% of the queue capacity – 10b = transitions above the 50% of the queue capacity – 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
08h	1	Fatal Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds – 00b = transitions from having no entries to having one or more entries – 01b = transitions above the 25% of the queue capacity – 10b = transitions above the 50% of the queue capacity – 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved

Table 4 Get Event Interrupt Policy Output payload

Set Event Interrupt Policy (Opcode 0103h): This command sets the interrupt method for the interrupts that are signaled by the device event. The same as with the “Get Event Interrupt Policy” some changes are necessary to extend this functionality.

The most important change is in order to enable interrupts to be generated not at the first event, but when the event queue has filled a percentage of the available queue. The options for the threshold can be controlled using the 2 bits available, from 0% which is the default in CXL to 75% in 25% increments.

We also need to set “Dynamic Capacity Event Log Interrupt Settings” to 00h since that event log is not implemented in RAS API.

Byte Offset	Length in Bytes	Description
00h	1	Informational Event Log Interrupt Settings: Specifies the settings for the interrupt when the information event log transitions from having no entries to having one or more entries. • Bits[1:0]: Interrupt Mode – 00b = No interrupts – 01b = MSI/MSI-X – 10b = FW Interrupt (EFN VDM) – 11b = Reserved • Bits[3:2]: Reserved • Bits[7:4]: FW Interrupt Message Number - Specifies the FW interrupt vector the device shall use to issue the firmware notification. Only valid if Interrupt Mode = FW Interrupt.

Open Compute Project • RAS API

01h	1	Warning Event Log Interrupt Settings: Specifies the settings for the interrupt when the warning event log transitions from having no entries to having one or more entries. • Bits[1:0]: Interrupt Mode — 00b = No interrupts — 01b = MSI/MSI-X — 10b = FW Interrupt (EFN VDM) — 11b = Reserved • Bits[3:2]: Reserved • Bits[7:4]: FW Interrupt Message Number - Specifies the FW interrupt vector the device shall use to issue the firmware notification. Only valid if Interrupt Mode = FW Interrupt.
02h	1	Failure Event Log Interrupt Settings: Specifies the settings for the interrupt when the failure event log transitions from having no entries to having one or more entries. • Bits[1:0]: Interrupt Mode — 00b = No interrupts — 01b = MSI/MSI-X — 10b = FW Interrupt (EFN VDM) — 11b = Reserved • Bits[3:2]: Reserved • Bits[7:4]: FW Interrupt Message Number - Specifies the FW interrupt vector the device shall use to issue the firmware notification. Only valid if Interrupt Mode = FW Interrupt.
03h	1	Fatal Event Log Interrupt Settings: Specifies the settings for the interrupt when the fatal event log transitions from having no entries to having one or more entries. • Bits[1:0]: Interrupt Mode — 00b = No interrupts — 01b = MSI/MSI-X — 10b = FW Interrupt (EFN VDM) — 11b = Reserved • Bits[3:2]: Reserved • Bits[7:4]: FW Interrupt Message Number - Specifies the FW interrupt vector the device shall use to issue the firmware notification. Only valid if Interrupt Mode = FW Interrupt.
04h	1	Reserved
05h	1	Informational Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries — 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
06h	1	Warning Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries — 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
07h	1	Failure Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries — 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved

08h	1	Fatal Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries — 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
-----	---	--

Table 5 Set Event Interrupt Policy Input payload

Get MCTP Event Interrupt Policy (Opcode 0104h): This command reads the setting for interrupts that are signaled by the device for components over MCTP. This also includes the events that are generate for the background operations in the MCTP based mailbox.

The most important change is in order to enable interrupt messages to be generated not at the first event, but when the event queue has filled a percentage of the available queue. The options for the threshold can be controlled using the 2 bits available, from 0% which is the default in CXL to 75% in 25% increments.

Another difference with the CXL spec is the bit 4 of the payload that represents the Dynamic Capacity Event Log, which is not implemented in the RAS API and will always return 0b.

Byte Offset	Length in Bytes	Description
00h	2	Event Interrupt Settings: Bitmask indicating whether event notifications are enabled (1) or disabled (0) for a particular event • Bit[0]: New uncleared Informational Event Log record(s) • Bit[1]: New uncleared Warning Event Log record(s) • Bit[2]: New uncleared Failure Event Log record(s) • Bit[3]: New uncleared Fatal Event Log record(s) • Bits[14:4]: Reserved • Bit[15]: Background Operation completed
02h	1	Informational Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries — 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
03h	1	Warning Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries — 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
04h	1	Failure Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries

		— 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved
05h	1	Fatal Event Log threshold Settings: Specifies the level at which if enabled the queue will send interrupts. • Bits[1:0]: Thresholds — 00b = transitions from having no entries to having one or more entries — 01b = transitions above the 25% of the queue capacity — 10b = transitions above the 50% of the queue capacity — 11b = transitions above the 75% of the queue capacity • Bits[7:2]: Reserved

Table 6 Get MCTP Event Interrupt Policy Input payload

Set MCTP Event Interrupt Policy (Opcode 0105h): This command is used to set the interrupt policy for components over the MCTP mailbox. The receiver captures the address of the requesting component to send the events to that address. The input payload is the same as the "Get MCTP Event Interrupt Policy".

Event Notification (Opcode 0106h): This command is the one used by the device to signal the interrupt to the driver; any message with this command to the device will be silently discarded. No changes are required from this opcode from the CXL original definition.

8.3 Timestamp

RAS API includes the commands for managing the timestamp that CXL has. These commands are built so that the software or management agent can set the timestamp on the device; removing the need for the device to have a Real Time Clock (RTC).

Get Timestamp (Opcode 0300h): Gets the timestamp from the device; if the timestamp has not been set it returns 0. Even though the timestamp is defined in nanoseconds, the hardware manufactures might not update the counter every nanosecond due to design constraints; so, it is recommended for hardware vendors to specify the frequency with which the device updates the timestamp on its records. No changes are required from this opcode from the CXL original definition.

Set Timestamp (Opcode 0301h): Sets the timestamp for the device; it is recommended to set it after every reset. The timestamp format in the input payload is "The number of unsigned nanoseconds that have elapsed since midnight, 01-Jan-1970, UTC." No changes are required from this opcode from the CXL original definition.

8.4 Trigger Action

Maintenance operation can be triggered using the commands explained here. These operations are available for some of the features as described in the "Features" in Chapter 6.

Not all the features have a triggering action, but those that do use the command explained in this section to trigger it.

Perform Maintenance (Opcode 0600h): This command executes a maintenance operation as described in the feature definition by class and subclass. The execution can create an immediate configuration change, data change, a log change, or a background operation triggering. The restriction and requirements for the maintenance operations are specify in each feature definition. No changes are required from this opcode from the CXL original definition. The table of features is expanded from the CXL definition as seen in Chapter 6.

8.5 Information and Status

The following set of commands are used to setup the mailbox mechanism and identify them. On some of this command is where the hardest changed for the opcodes are required.

Identify (Opcode 0001h): This command is used for the MCTP mailbox to determine if the mailbox is ready to receive commands and the size of the commands it can receive. If the mailbox is not ready, it should return a "Retry Required" code. For the output payload the message is simplified since this is not a CXL device and therefore has no need of the Component Type. The output payload should look like this:

Byte Offset	Length in Bytes	Description
00h	2	PCIe Vendor ID: Identifies the manufacturer of the component, as defined in PCIe Base Specification
02h	2	PCIe Device ID: Identifier for this particular component assigned by the vendor, as defined in PCIe Base Specification
04h	2	PCIe Subsystem Vendor ID: Identifies the manufacturer of the subsystem, as defined in PCIe Base Specification
06h	2	PCIe Subsystem ID: Identifier for this particular subsystem assigned by the vendor, as defined in PCIe Base Specification
08h	8	Device Serial Number: Unique identifier for this device, as defined in the Device Serial Number Extended Capability in PCIe Base Specification
16h	1	Maximum Supported Message Size: The maximum supported size of the full message body in bytes for any request sent to this component, expressed as 2^n . The minimum supported size is 256 bytes ($n=8$) and the maximum supported size is 1 MB ($n=20$). This field is used by the caller to limit the Message Payload size such that the size of the Message Body does not exceed the capabilities of the component. The component shall discard any received messages that exceed the maximum size advertised in this field in a manner that prevents any internal receiver hardware errors. The component shall return a response message with the 'Invalid Payload Length' return code for all received request messages that exceed the maximum size advertised in this field. The CXL specification guarantees that the size of the Identify Output Payload shall never exceed 244 Bytes (256 – 12 Bytes, the combined size of the fields preceding Message Payload).

Table 7 Identify Output Payload

Background Operation Status (Opcode 0002h): This command is used by the MCTP mailbox to determine the progress and status of a background operation. The MMIO mailbox has registers that enable this functionality, so it doesn't need this command. No changes are required from this opcode from the CXL original definition.

Get Response Message Limit (Opcode 0003h): This command is used to obtain the maximum message limit used by the MCTP mailbox. No changes are required from this opcode from the CXL original definition.

Set Response Message Limit (Opcode 0004h): This command sets the maximum size of the full message body for the MCTP mailbox. The return payload has the maximum size that has been set on the device that could be lower than the support size of the agent. No changes are required from this opcode from the CXL original definition.

8.6 New OPCODES

A new set of opcodes has been defined to help with the arbitration between different mailboxes trying to execute different RAS action on the device.

Group	Command	MMIO Mailbox	MCTP Mailbox
Information and Status	Request feature ownership	Yes	Yes
	Release feature ownership	Yes	Yes

Request Feature Ownership (Opcode 0015h): This opcode requests the control of a RAS feature or an event log.

The mailbox that issues this opcode can request an available feature or event log, if the design of the platform has enabled that mailbox to own it (For further details on the usage see Section 4.2.1).

The input payload for this opcode is as define in the following table. The opcode can be used to request the ownership of a feature (setting Flags. Bit [1] to 1) or an event log (setting that same bitt to 0). When the Query flag is set the opcode will not request the ownership, but query about its availability.

Byte Offset	Length in Bytes	Description
00h	1	Flags • Bit [0] Query flag If set, the Device will only check if the mailbox is available. • Bit [1] Feature/ Event log flag If set, the request is for a feature, otherwise for an event log. • Bit [7:2] Reserved.
01h	10	Feature Identifier: UUID representing the Feature identifier for which data is being retrieved. 0h if the request is for an event log.
0Bh	1	Event Log Identifier: Determines the event log being requested: • 0h: None • 1h: Informational Event • 2h: Warning Event • 3h: Failure Event • 4h: Fatal Event • Rest – Reserved

Table 8 Input payload for Request Feature Ownership

The output of the Request Feature Ownership can be interpreted using the following table. The most important highlight is when the query flag has been enabled that the return code is interpreted as the mailbox being available as opposed to the mailbox's ownership being set to a new owner.

Value	Definition	Query mailbox flag = 0	Query mailbox flag = 0
0000h	Success	Feature ownership successfully claimed by this mailbox	Feature ownership is available to be claimed
0003h	Unsupported	Command is not supported	Command is not supported
001Dh	Resources Exhausted:	Feature ownership failed because feature is claimed by a different mailbox	Feature ownership is NOT available to be claimed

Table 9 Output of Request Feature ownership

Release Feature Ownership (Opcode 0016h): This opcode releases the control of a RAS feature or an event log.

The mailbox that issues this opcode can release a feature or event log to make it available to another mailbox in the system. The functioning of this opcode is analogous to the request feature ownership, with the exception that no query needs to be implemented.

Byte Offset	Length in Bytes	Description
00h	1	Flags • Bit [0] Reserved • Bit [1] Feature/ Event log flag If set, the request is for a feature, otherwise for an event log. • Bit [7:2] Reserved.
01h	10	Feature Identifier: UUID representing the Feature identifier for which data is being retrieved. 0h if the request is for an event log.
0Bh	1	Event Log Identifier: Determines the event log being requested: • 0h: None • 1h: Informational Event • 2h: Warning Event • 3h: Failure Event • 4h: Fatal Event • Rest – Reserved

Table 10 Input payload for Release Feature Ownership

The output from this opcode is also a simplify equivalent to the request opcode. The following table shows the expected outputs from the opcode.

Value	Definition	Output
0000h	Success	Feature ownership successfully unclaimed by this mailbox
0003h	Unsupported	Command is not supported
001Dh	Resources Exhausted:	Feature not owned by this mailbox

Table 11 Output of Release Feature Ownership

9. Features

There are several features that comprise the RAS API.

In this chapter we will go through all of them. Some features for memory are being leverage from the CXL spec and the reference to the document will be shown, as well as some light complementary information for the feature. For the features that are completely new a more detailed description is provided.

9.1 Feature use modes

Features can have two use cases: autonomous (also called device initiated) or assisted mode (also called host- initiated), These use cases are optional, but at least one needs to be supported to trigger a RAS action. Features that do not have RAS actions (does that work only with configuration) will not support either mode but will work through their attributes.

The following figure depicts the events for an OS managed system that has the auto memory sparing feature set to autonomous mode.

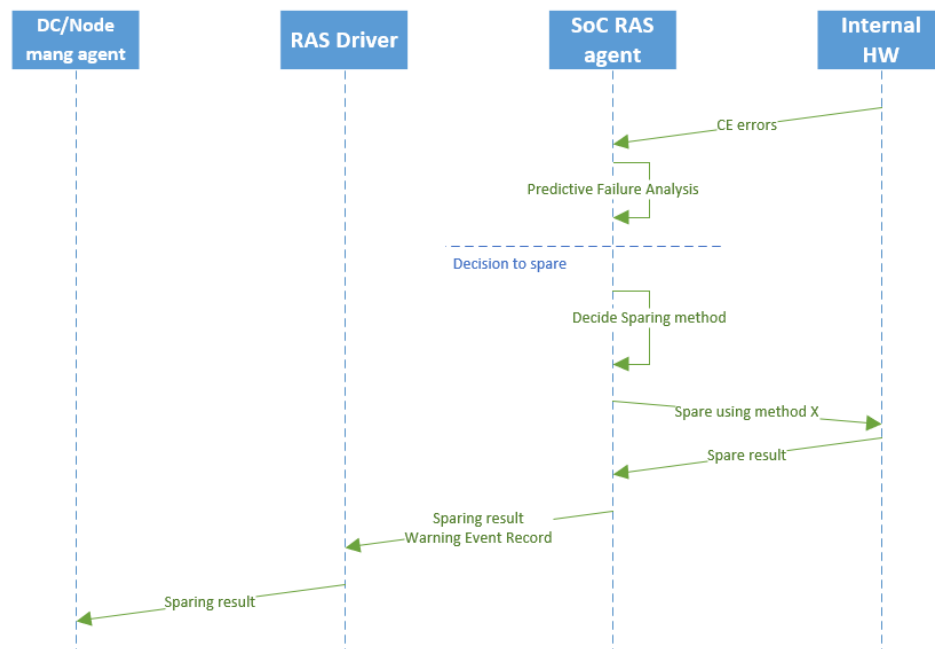


Figure 13 Auto Memory Sparing - Autonomous mode

To provide some contrast in the following figure the Auto Memory sparing is shown in assisted mode, but also using the OS as the management software.

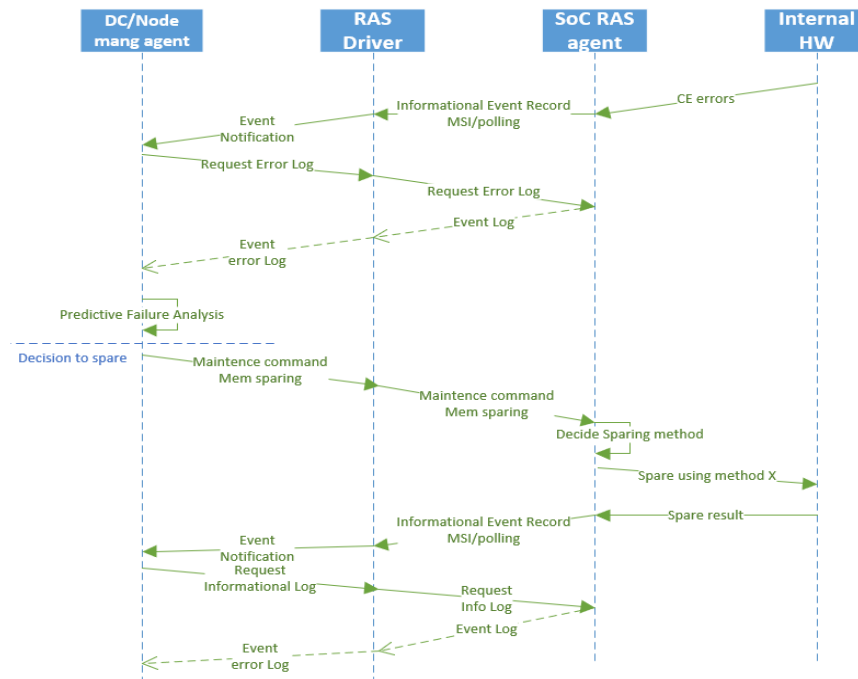


Figure 14 Auto Memory Sparing - Assisted mode

Features have a common set of attributes that are described briefly in Chapter 5.1. It is important to note that not all features can support autonomous or assisted mode; some feature will have only one mode that must be used. Some features cannot be controlled by the host and can only be configured by it, so those features are autonomous only; an example of this is demand scrub for some systems that can't be turned off in any way. The opposite example also exists; some features that cannot be autonomous that will always require the host triggering mechanism; an example of this is Sparing using methods that have a long execution time.

Features work through the mailbox as explained before, so it is recommended to have features that take more than 2s to execute in the background using the background command capabilities of the mailbox. If those features interrupt the normal operation of the device, then it is advice for those features not to have an autonomous mode. This prevents the system from having lock outs of resources or changed in capabilities outside the host's control.

9.2 List of RAS features

The following chapters the list of all the RAS features proposed in this RAS API standard. Notice that they are not necessarily subdivided by their corresponding IP, but a flat list with operation class and subclass (as inherited an extended from CXL). The list is subdivided in chapters so each feature can be properly explained. For CXL adopted features the details of the payloads and such should always reference the CXL specification. This specification highlights

the differences with CXL specification as well as which features are new to RAS API that are not defined in CXL.

9.2.1 Memory features

Table 12 shows the Memory RAS features that are going to be discussed over the next sub chapters. Notice that some of these features are in ECN from recent changes to the CXL specifications and others are new to the RAS API and will be inserted in the CXL specification soon. This only happens with the memory features since the CXL specification refers to type III devices that are memory devices and therefore need the same RAS features for memory as we would expect in other platform devices.

Feature Name	Feature UUID
Soft PPR	892ba475-fad8-474e-9d3e-692c917568bb
Hard PPR	80ea4521-786f-4127-afb1-ec7459fb0e24
DDR Memory Enhanced Sparing	Placeholder

Table 12 Memory RAS features

In addition, some of the features have maintenance capabilities that can be triggered using the maintenance command. A list of the maintenance capabilities is shown on Table 13.

Maintenance Operation Class		Maintenance Operation Subclass	
Value	Description	Value	Description
00h	No operation	00h	No operation
01h	Sparing	00h	Soft PPR
		01h	Hard PPR
		Others	Reserved
		All	Reserved
05h - AFh	Reserved (CXL)		

Table 13 Memory RAS maintenance capabilities

Soft PPR

This feature refers to the JEDEC defined sPPR as a “way to quickly, but temporarily, repair one row address per Bank Group...” This feature should retain the soft repair information if the power supply remains within its operating range and there is no DRAM reset. When either of these conditions happen, the DRAM will revert to its un-repaired state. This feature usually uses the same resources as hard PPR, so care must be taken of when the resources are exhausted.

There is no change to its feature from the CXL definition.

Hard PPR

This feature refers to the JEDEC defined as hPPR. In chapter 4.29 of the JEDEC standard No 79-5 the PPR is defined as a Fail Row address repair which allows a simple and easy repair method in a system. The Hard PPR is permanent and according to the specification the repairs done this way are permanent and cannot be switched back to their un-fused state once they are programmed.

There is no change to its feature from the CXL definition.

DDR Memory Enhanced Sparing

This feature will be developed for the next revision of the spec.

9.2.2 Link features

The link RAS features propose shown in the following table: TBD

Maintenance Operation Class		Maintenance Operation Subclass	
Value	Description	Value	Description
C0h		00h	

9.2.3 Core features

The core RAS features includes features that are used to control the detection or correction of core errors in CPU as well as GPUs.

Maintenance Operation Class		Maintenance Operation Subclass	
Value	Description	Value	Description
D0h	Core RAS features	00h	Self-test
		01h	Core Repair

9.2.4 Error injection

These are the error injection features normalized in the RAS API:

Maintenance Operation Class		Maintenance Operation Subclass	
Value	Description	Value	Description
E0h	Error Injection	00h	Memory Error injection
		01h	Core Error injection

10. Error logs

Error logs in the RAS API are handled using the event log mechanism that the CXL specification has. The methodology consists of 4 queue of errors that are separated by severity instead of IP. The queue themselves exist for each mailbox that is implemented in the device. This eliminates the race conditions that arise from several devices reading the same information from the hardware.

It is also important to highlight that the queue have no overwrite rules, but instead use the overflow flag and a couple of register to determine the sampling that the errors are getting at this particular severity.

For the opcodes and more information on how to set up the error logs please refer to Chapter 5.2.

10.1 Event record formats

The different event record formats can be used to describe different types of errors. The idea of the RAS API standardization is to determine which are the best error formats for the specific errors described for each section.

This API will leverage the common event record format for the records of all their types. The specifics of each error will be discussed in later chapters.

The common area for the event records has several important fields that are worth highlighting for this RAS API:

The **event record identifier (UUID)** determines the type of event records that is being read. It will determine all the fields that are describing the error according to the spec. The **event record severity** must match the log from which this record was read, and it shows the severity of the error. The other flags help understand better the expected outcome from the failure: **permanent condition, maintenance needed, performance degraded, or hardware replacement needed**.

Finally, it is important to note that every error record should have the timestamp of when it happened. This field is interesting for RAS since it allows for debugging and Root Cause Analysis (RCA); since many errors are usually a byproduct of previous errors, and determine their order is a big part of how to untangle the RCA.

10.1.1 DRAM Memory error record

The memory error record uses the DRAM format in CXL. In RAS API we leverage the full record for the DRAM event record in the CXL 2.0 spec.

For this spec it is important to note the use of the memory address since this will allow the assisted mode to determine the faulty addresses and eventually request action on those. The Physical Address field has the DPA address that is particular to the device itself. Also, among the field that are populated are the channel, rank, nibble mask, bank group, bank, row, column. This information should help determine the exact bits that flip in the memory failure.

10.1.2 CPER error record

RAS API requires to extend the memory errors that exist in CXL to more generic errors that can be represented for RAS agents throughout the platform. To do this the Common Platform Error Record (CPER) as defined by the UEFI standard is normalized here as a new error record type.

Byte Offset	Length in Bytes	Description
00h	30h	Common Event Record: See corresponding common event record fields defined in CXL spec 2.0 Section 8.2.9.2.1. The Event Record Identifier field shall be set to CPER UUID "79499ac0-40d3-44c9-832e-d9ea3d38c12f" representing the format.
30h	50h	CPER record

Table 14 CPER Event Record

This record format is easy to adopt from the hardware vendors as well as easy to integrate of the datacenter owners since it has a lot of adoption in the market.

11. References (recommended)

[1] T. Islam and D. Manivannan, "Predicting Application Failure in Cloud: A Machine Learning Approach," 2017 IEEE International Conference on Cognitive Computing (ICCC), Honolulu, HI, USA, 2017, pp. 24-31, doi: 10.1109/IEEE.ICCC.2017.11.

12. Appendix A - Checklist for IC approval of this Specification (to be completed by contributor(s) of this Spec)

Complete all the checklist items in the table with links to the section where it is described in this spec or an external document .

Item	Status or Details	Link to detailed explanation
Is this contribution entered into the OCP Contribution Portal?	Yes or No	If no, please state reason.
Was it approved in the OCP Contribution Portal?	Yes or No	If no, please state reason.
Is there a Supplier(s) that is building a product based on this Spec? (Supplier must be an OCP Solution Provider)	No	
Will Supplier(s) have the product available for GENERAL AVAILABILITY within 120 days?	No	If more time is required, please state the timeline and reason for extension request. Please have each Supplier fill out Appendix B.

13.

14. Appendix B-__ <supplier name> - OCP Supplier Information and Hardware Product Recognition Checklist

(to be provided by each supplier seeking OCP recognition for a Hardware Product based on this specification)

Company:

Contact Info:

Product Name:

Product SKU#:

Link to Product Landing Page:

The following is needed for OCP hardware product recognition:

For OCP Inspired™

- All Suppliers must be a Silver, Gold or Platinum Member.
- Declare product is 100% compliant with specification
- Complete the [OCP Inspired™ Product Recognition Checklist](#), which includes hardware management conformance checks and security profile.

For OCP Accepted™

- All Suppliers must be an OCP Member. All corporate membership levels are eligible.
- Complete the [OCP Accepted™ Product Recognition Checklist](#), which includes hardware management conformance checks, security profile and open system firmware conformance checks.
- Submit a design package meeting [OCP Hardware Design Guideline Contribution Checklist](#) (if not already submitted by the contributor). If already submitted, declare the product is 100% compliant with the design package.
- Submit a firmware package including a firmware image, build scripts, documentation, test results and a tool that verifies modifications
- Submit the BMC source code, if applicable to product type

Please complete the OCP Inspired™ Product Recognition Submission Checklist or OCP Accepted™ Product Recognition Checklist and the following table.

Item	Details	Links
Which product recognition?	OCP Accepted™ or OCP Inspired™	Provide link for the appropriate Product Checklist
If OCP Accepted™, who provided the Design Package?		Link to OCP Contribution Database
Where can a potential adopter purchase the product?		Link to OCP Marketplace

15.

16. Appendix C - ACPI table example for in band CPU RAS agent discovery

Field	Byte Length	Byte Offset	Description
Signature	4	0	'XXXX'. Find an unused Signature
Length	4	4	Length, in bytes, of the description table including the length of the RAS API Configuration structures.
Revision	1	8	Must be 1.
Checksum	1	9	Entire table must sum to zero.
OEMID	6	10	OEM ID.
OEM Table ID	8	16	The Table ID is the manufacturer model ID.
OEM Revision	4	24	OEM Revision of the Table for OEM Table ID
Creator ID	4	28	Vendor ID of utility that created the table.
Creator Revision	4	32	Revision of utility that created the table.
Reserved	4	36	Reserved (0)
RAS Configuration Unit Structure	-	40	A list of RAS API Configuration Structure.

Table 15 ACPI In bound discovery table

Field	Byte Length	Byte Offset	Description
Type	1	0	00 -RAS API configuration structure
Reserved	1	1	Reserved
Length	2	2	Length of the entire RAS API Configuration structure, including the header.
Start x2APIC ID	4	4	The lowest APIC ID value to which this structure applies.
End x2APIC ID	4	8	The highest APIC ID value to which this structure applies.
x2APIC ID mask	2	12	Mask for APIC ID's to which this structure applies.
MC Banks	4	14	A bitmap representing the MC banks associated with the APIC IDs in {Start APIC ID, End APIC ID} range to which this structure applies
Base	12	18	ACPI Generic Address Space (GAS) structure that points to the RAS API mailbox base address
Length	4	30	Length of the RAS API mailbox
Interrupt Message number	1	34	Number of the interrupt use for this RAS API mailbox
Handle Count	1	35	Number of SMBIOS handles in the array below
Reserved	2	36	Reserved
SMBIOS Handles	4 * n	38	Enumerates the SMBIOS handles associated with the Memory DIMMs that are controlled by this Unit. Software uses the component identifier field in the Event Record to index into this table and locate the SMBIOS entry corresponding to the FRU in error.

Table 16 RAS API Configuration Structure