



OPEN

Compute Project

RAS API Specification Version 0.9

Dec 14, 2025

Version History

Note, refer to the OCP Contribution Versions, Revisions and Errata best practices documentation. Generally speaking, Versions and revisions are made to this document and logged here. Errata is a separate document such that the contribution specification document it refers to was not revised. Ex: Version 1, Errata E1

Date	Version	Revision	Authors	Description
February 2023	0.8	1	Intel Corporation Google LLC	Initial specification released to OCP.
December 2025	0.9	1	AMD, Ampere, Arm, ByteDance, Dell, Google, Intel, Meta, Microsoft, Oracle, PureStorage, Synposys, Yogesh Varma (independent)	Interim release to reflect change of direction, to give visibility to industry partners. Revised with CPER+CPAD requirements. Removed sections that became inapplicable due to redefinition of the abstraction layer.

Current Template Version (Reviewed)

3 Layer (Base, Design and Product) Specification Template V1.0.0
Effective January 2024

Table of Contents

Version History.....	2
Current Template Version (Reviewed).....	2
1. License.....	4
2. Acknowledgements.....	6
3. Compliance with OCP Tenets.....	7
3.1. Openness.....	7
3.2. Efficiency.....	7
3.3. Impact.....	7
3.4. Sustainability.....	7
4. Scope.....	8
5. Terminology.....	8
6. Requirements.....	9
7. Overview.....	9
8. Base Specifications.....	11
8.1. General guidelines.....	11
8.2. Abstraction Interface.....	11
8.2.1. Discovery.....	12
8.2.1.1. Commands for Discovery.....	12
8.2.2. Configuration.....	13
8.2.2.1. Commands for Configuration.....	13
8.2.3. Error Reporting Queues.....	13
8.2.3.1. Commands for Error Queue Management.....	14
8.2.3.2. Event Reporting Queue.....	14
8.2.4. Action Queue.....	15
8.2.4.1. Commands for Action Queue Management.....	15
8.3. Data Structures.....	16
8.3.1. Common Platform Error Record (CPER).....	16
8.3.2. Common Platform Action Descriptor (CPAD).....	18
8.3.2.1. CPAD General Format.....	18
8.3.3. CPAD Record Header.....	19
8.3.4. CPAD Section Descriptor.....	21
8.3.5. CPAD Section Body.....	23
9. Glossary of Terms.....	24
10. References.....	24

1. License

Contributions to this Specification are made under the terms and conditions set forth in **Modified Open Web Foundation Agreement 0.9 (OWFa 0.9)**. (As of October 16, 2024) (“Contribution License”) by:

Alibaba
AMD
Ampere
Arm
ByteDance
Dell
Insyde
Intel
Inventec
Google
Meta
Microsoft
Nvidia
Oracle
Pure Storage
SKHynix
Synopsys
Tencent
Yogesh Varma (independent)

Usage of this Specification is governed by the terms and conditions set forth in **Modified OWFa 0.9 Final Specification Agreement (FSA)** (As of October 16, 2024) (“**Specification License**”).

You can review the applicable Specification License(s) referenced above by the contributors to this Specification on the OCP website at <https://www.opencompute.org/contributions/templates-agreements>.

For actual executed copies of either agreement, please contact OCP directly.

Notes:

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. Acknowledgements

The Contributors of this Specification would like to acknowledge the following for their feedback:

List all companies or individuals who may have assisted you with the specification by providing feedback and suggestions but did not provide any IP.

Name of the CLA Participant Company	Names of the Contributing Individuals from the CLA Participant Company
AMD	Vilas Sridharan Sarathy Jaykumar
Ampere	Seb Hily Tyler Baicar
Arm	Dean Liberty Jose Marinho
ByteDance	Terry Gong
Independent	Yogesh Varma (Co-lead OCP RAS API)
Intel	Shubhada Pugaonkar
Google	Sean Malley
Meta	Anil Agrawal Sukay Luhadia
Microsoft	Carlos Valin Drew Walton (Co-lead OCP RAS API) Rama Bhimanadhuni
Oracle	Aimer Bhat
Pure Storage	Meeta Saggi
Synopsys	Durga Prasad Sahoo

3. Compliance with OCP Tenets

3.1. 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 is expected to allow OEMs and CSPs to develop their management systems around this spec.

3.2. Efficiency

Current RAS implementations are proprietary and a great majority of them require firmware expertise to get deployed. The simplification and hardware abstraction that are achieved through RAS API enable smaller teams to experiment and deploy RAS features easily and quickly. 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, creating yet another efficiency for end customers.

3.3. 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.

3.4. Sustainability

A paper by Google [\[1\]](#) analyzed and projected the number of resources wasted due to job failure and relaunch. 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 analyses 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.

4. Scope

The scope of this document is to define the RAS API data exchange and communication protocol that is transport agnostic. RAS API is an optional interface for components and defines mechanisms for managing RAS usages. This document covers the following key aspects:

1. The RAS API abstraction interface
2. Commands and responses to interact with the RAS API abstraction
3. Standardized data structures used as payloads.

Additionally, this document may also describe implementation considerations related to particular features.

Out of Scope

The requirements defined in this specification apply when a system implements a particular capability in a manner that conforms with this specification. However, this specification does not specify whether a given system is required to implement that capability.

This specification is not intended to address system-level platform requirements. However, parts of this specification may rely on other relevant specifications or related initiatives. The following references are particularly relevant:

1. [OCP Hardware Fault Management Infrastructure Requirements](#)
2. [OCP Fleet Memory Fault Management Framework Requirements](#)

5. Terminology

Term	Description
RAS API	The data exchange and communication protocol for discovery, configuration, and interactions between RAS API agents
RAS API Agent	Entity that understands the RAS API protocol. A RAS API Agent could be either an Endpoint (EP) or an Initiator
RAS API Endpoint	Agent that exposes RAS API abstraction and services the API requests
RAS API Initiator	Agent that initiates the RAS API requests
Transport Layer	The underlying transport and/or data model used to exchange RAS API commands and responses

6. Requirements

This section outlines the foundational requirements necessary to achieve the objectives of the RAS API initiative. These requirements are derived from common pain-points and desired enhancements generally accepted within the RAS community. Furthermore, they are consistent with the requirements outlined in the [OCP Hardware Fault Management Infrastructure Requirements](#) document produced by the OCP Hardware Fault Management team.

1. The OCP RAS API specification shall be component and interface agnostic, meaning it can be applied to all components (e.g. CPU, GPU, Accelerators) and usage models (e.g. In-band and/or Out-of-Band).
2. The RAS API abstraction should enable Server Fleet Operators to write a data center software interface that remains consistent across new hardware generations and suppliers, provided they comply with this specification.
3. The RAS API shall enable dynamic discovery, enumeration and configuration of supported RAS capabilities.
4. The RAS API shall mandate a standardized error format and queuing system, optimizing error data collection for large-scale operations.
5. The RAS API shall support requests for actions to endpoints in a standardized manner. Actions may be associated with error records.
6. The RAS API definition shall include provisions for supporting proprietary functions in a manner that allows suppliers to innovate without necessitating modifications to Server Fleet Operator infrastructure.
7. The RAS API definition shall include provisions for version control and maintain backwards compatibility with prior versions.
8. The RAS API shall enable configuring, logging errors from, and sending platform actions to physical Field Replaceable Units (FRUs).

7. Overview

The RAS API provides an abstraction layer that standardizes fault management interfaces. The RAS API consists of RAS API Endpoints and RAS API Initiators as shown below:

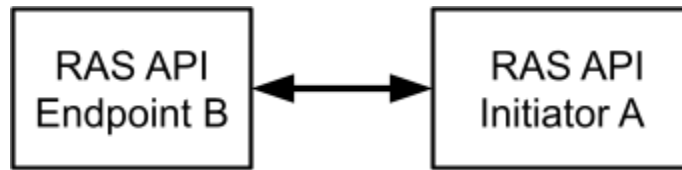


Figure 1: RAS API Network

The arrow is RAS API Protocol mapping to some implementation specific transport layer (e.g. PCIE, PLDM, etc.) The RAS API Protocol is agnostic to the transport layer and the mapping of the RAS API protocol to the transport layer is not defined in this document.

The RAS API networks can be arranged in up to 2 deep hierarchical topologies as shown in the following example:

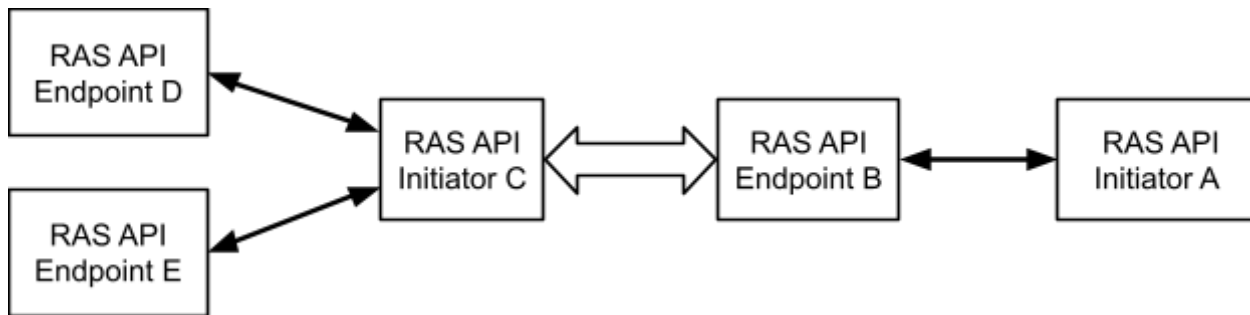


Figure 2: Example Hierarchical RAS API Network Topology

One key use case for such a 2 deep topology reflects the [OCP Hardware Fault Management Infrastructure Requirements](#) document where endpoints D and E are implemented in silicon such as CPUs and IO devices and where initiator C and Endpoint B may represent an Out-of-Band Management Agent (OMA). In such a use case Initiator A may act as a collector that feeds error logs to error analysis tools and sends CPADs from the analysis tools back to the RAS API endpoints. The thick arrow between Endpoint B and Initiator C is not RAS API protocol and not presently defined in the specification.

The RAS API endpoints provide a standard way for management agents to discover them, configure them, collect error data from them, and perform platform actions on them.

RAS API Endpoints can be connected in a hierarchical manner as shown above. The arrows between components represent system management transport layers to which the RAS API has been mapped. The RAS API operations themselves are transport layer agnostic. The OCP RAS API workstream will coordinate with the appropriate standards bodies, e.g. DMTF, ACPI, to map RAS API operations to management transport layers.

Errors are collected from RAS API endpoints using the Common Platform Error Record (CPER) format via Initiators that execute RAS API commands. The Initiator triggers actions on RAS API

endpoints by sending them Common Platform Action Descriptors (CPADs). A CPAD may also describe manual interventions such as replacing parts.

The RAS API also defines how fields in CPERs and CPADs descriptors are used to ensure that CPADs can be delivered to the correct RAS API endpoints. CPER and CPAD section bodies may contain data in a proprietary format and CPAD descriptors support both standard and proprietary actions.

This specification provides a set of minimum requirements for silicon vendors and data structures, opcodes, and commands for various use cases. Examples may be provided to clarify the collective needs related to HW fault management of the data center fleet. These guidelines will encourage vendors to lead and refine protocol specifications that fulfill these requirements. Consequently, software drivers developed in accordance with these enhancements will comply with both the RAS requirements and the software interface definitions outlined in this document.

8. Base Specifications

This section contains the normative requirements of the OCP RAS API specification. All statements and requirements described herein use normative language (such as "shall," "must," and "should") to define the mandatory behaviors, interfaces, and data structures that must be implemented for compliance. Implementers are expected to adhere to these requirements to ensure interoperability and conformance with this specification.

8.1. General guidelines

To ensure the broadest impact and facilitate a unified approach to RAS features within the data center ecosystem, it is critical that all specifications, including this OCP RAS specification, are shared openly. This document is being crafted with transparency in the open-source domain, and it is imperative that any companion specifications required for a comprehensive solution follow suit. By contributing these additional protocol layer specifications to open-source projects, we can extend the benefits beyond a proprietary framework, allowing for widespread adoption and collaboration across multiple vendors.

8.2. Abstraction Interface

RAS API Endpoints shall expose an abstraction interface comprising the following components:

- Discovery
- Configuration
- Independent sets of logical error reporting queues per supported interface instance
- Independent logical action queues per supported interface instance
- Independent logical event queues per supported interface instance

The following diagram illustrates this concept.

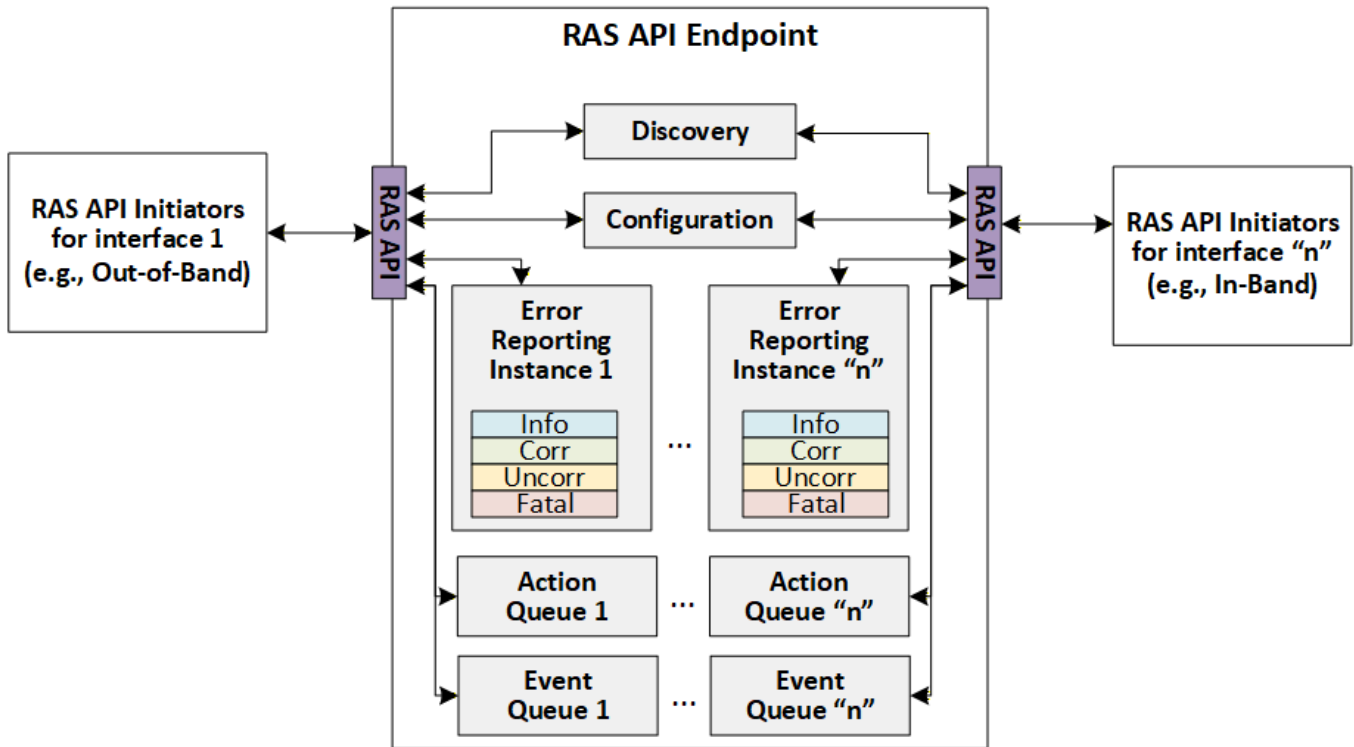


Figure 3: High-level diagram of RAS API Endpoint abstraction

The **key use case** targeted by this specification is to expose two sets of separate logical error reporting and action queues, intended for dedicated access and management by out-of-band and in-band RAS API initiators (i.e. “n” = 2). However, the abstraction requirements are specified to accommodate potential use cases that may necessitate more than two sets of logical queues.

The following sections define each abstraction component in more detail.

8.2.1. Discovery

The RAS API Endpoint shall provide functionality to enable programmatic discovery of RAS API support level and the capabilities supported by the Endpoint.

8.2.1.1. Commands for Discovery

The RAS API supports discovery commands that allow an initiator to discover endpoints, including their type and identity. The discovery commands provide information necessary for a server operator’s fault management infrastructure to configure endpoints, collect error data from the

endpoints, analyze that data, and send platform actions back to the endpoint. These discovery commands will be described in the next version of the specification.

8.2.2. Configuration

The RAS API Endpoint shall provide functionality to configure the endpoint.

8.2.2.1. Commands for Configuration

The following table summarizes the functionality enabled by the command set for RAS API configuration capabilities.

Function	Minimum Response	Command
Retrieve basic information about each RAS API EP's capabilities	Config data	Will be defined in future revision
Enable RAS API initiators to configure EP's settings.	Completion code	

More details, including potential conditional and optional functionality, will be provided in future releases of this specification.

8.2.3. Error Reporting Queues

The specification mandates separate sets of queue representation for error reporting to each RAS API interface to prevent any interaction between them. Each set contains four severity-based queues. Error records are categorized into the queue that matches its severity.

The severity field in the CPER header (corresponding to the most severe error section in the CPER) shall be used to assign error records to their corresponding error queue.

The error queues are defined as follows:

- **Fatal Error Queue:** For error logs containing critical events that enable Root Cause Analysis (RCA) of fatal issues.
- **Recoverable (also called non-fatal uncorrected):** For error logs necessitating immediate management entity intervention, often part of recovery processes.
- **Corrected Error Queue:** For error logs that may require future action, like corrected errors, which inform Predictive Failure Analysis (PFA) but are not immediately actionable.
- **Informational Queue:** For error logs that may require future actions, including repair actions.

Hardware Fault Management (HWFM) requirements map various server platform error types to UEFI CPER defined error severities.

While hardware vendors may determine queue sizes, it is advised that corrected and informational queues have greater capacity than those for uncorrected non-fatal and uncorrected fatal errors, reflecting their expected frequency. Additionally, queues must preserve error records through warm resets to support post-reset error harvesting for Root Cause Analysis (RCA).

Each queue should independently manage its interrupt policy, allowing the RAS API initiator to adjust between polling and interrupt generation in real-time. As an optional feature, queues may offer configurable interrupt thresholds (e.g., first error, half full, 90% full) to aid in error storm management. During such storms, the driver's ability to modify interrupt behavior is crucial.

8.2.3.1. Commands for Error Queue Management

This section outlines a set of RAS API queue management operations that enable initiators to discover, configure, monitor, retrieve, and clear error queues in a policy-driven manner.

More details, including potential conditional and optional functionality, will be provided in future releases of this specification.

Function	Minimum Response	Command
Retrieve queue related configuration	Config data	Will be defined in next revision
Program queue related configuration	Completion code	
Collect Event log(s)	CPER logs	
Clear the event log(s) that were collected	Completion code	

8.2.3.2. Event Reporting Queue

The RAS API Endpoint shall expose a logical queue used for events that are not categorized as errors. The key use case is to report results from platform actions requested through the action queue (i.e. CPADs). The maximum depth of this queue is implementation specific.

These are not intended to be frequent, non-error events like power state changes.

For example, passing a RAS action's status back to the initiator allows an error analysis tool to determine if an action was successful or not. This prevents analysis tools from proposing the same failing action repeatedly. It allows the analysis tools to propose a different action, e.g. if a self-repair action fails, analyzers might suggest replacing a part.

The results from platform actions shall allow association to corresponding CPADs.

The events are CPER formatted and are intended to be collected by the same initiator that collects error logs. The CPER encoding shall be defined in future versions of this document.

8.2.4. Action Queue

The RAS API Endpoint shall expose a logical queue for requesting execution of actions. The minimum depth of this queue shall be 1. The maximum depth of this queue is implementation specific.

The data structure used for each action request shall follow the Common Platform Action Descriptor (CPAD) format defined in Section [8.3.2](#) and the actions will be described in that format.

More details will be provided in future releases of this specification.

8.2.4.1. Commands for Action Queue Management

The following table summarizes the functionality that must be enabled by the command set to allow RAS API Initiators to interact with each Endpoint's Action queue.

Function	Min Response	Command
Retrieve action queue settings	Config Data	Will be defined in next revision
Configure action queue settings	Completion code	
Retrieve list of actions that are supported at the EP	Supported Action list	
Request execution of actions by sending CPAD	Completion code	
Retrieve status of action queue	Status Data	

More details, including potential conditional and optional functionality, will be provided in future releases of this specification.

8.3. Data Structures

This section defines the data structures standardized for use as RAS payloads by this specification. The following table summarizes it.

Category	Data Structure used
Error Queue record format	Common Platform Error Record (CPER)
Action queue record format	Common Platform Action Descriptor (CPAD)

Following diagram shows Example IDs mapped to various components:

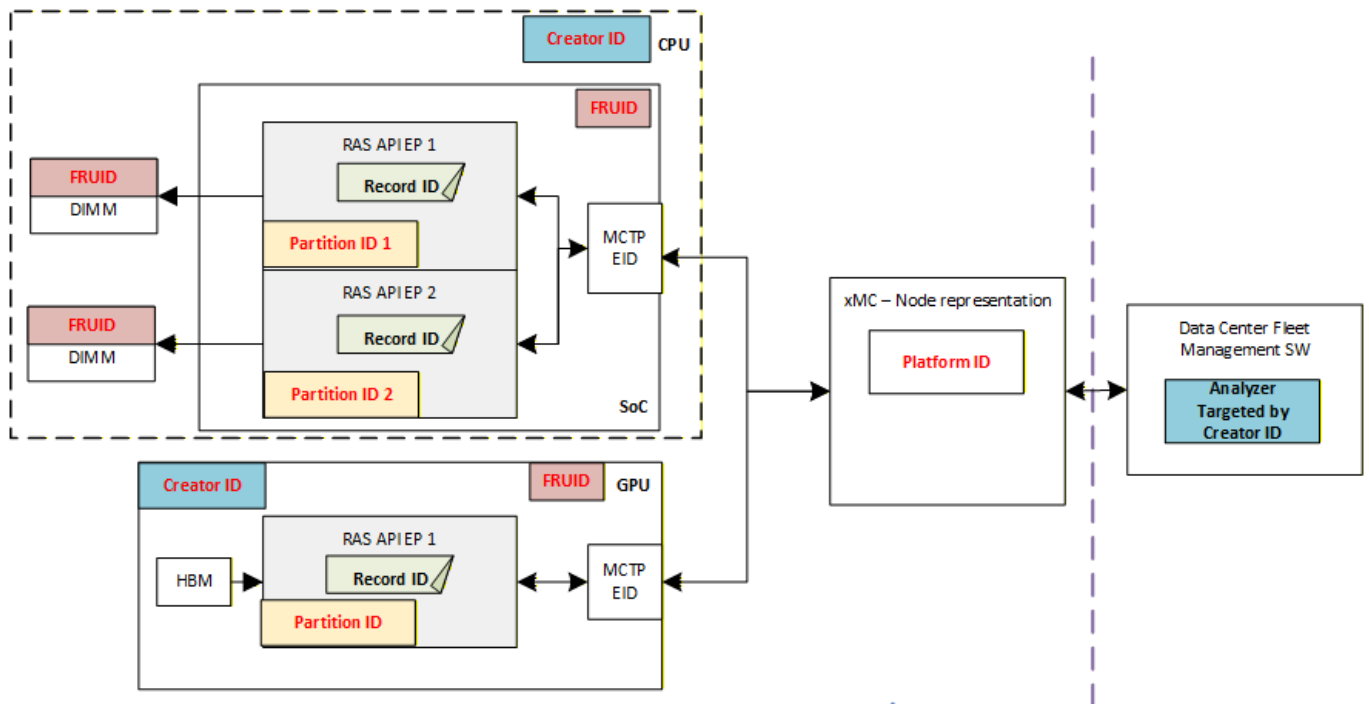


Figure 4: Example of RAS API IDs mapped to various components

8.3.1. Common Platform Error Record (CPER)

This specification mandates error records to adhere to the Common Platform Error Record (CPER) format, as defined in the [UEFI specification - Appendix N](#).

In addition to complying with the UEFI specification, RAS API Endpoints must populate CPER fields with values enabling the end-to-end flows and use cases defined in [OCP Hardware Fault Management Infrastructure Requirements](#):

1. Header.timestamp field contains the local time in BCD format. Refer to CPER format for additional field description.
 - a. The CPER specification states, “The timestamp correlates to the time when the error information was collected by the system software and may not necessarily represent the time of the error event.”
 - b. The expected use case is to track error records relative to each other for time-series analysis and to correlate with other data.
2. Header.PlatformID field is a platform-assigned GUID that uniquely identifies the platform. How the platform generates this value is outside the scope of this specification.
 - a. The PlatformID must allow tracking of error history at the platform level within the fleet.
 - b. CPERs from all endpoints in a platform shall have the same PlatformID.
3. Header.PartitionID field is a silicon-vendor-assigned GUID that uniquely identifies the RAS API Endpoint within the platform. How the silicon-vendor generates this value is outside the scope of this specification.
 - a. The PartitionID value is used with the PlatformID value to identify the endpoint that created the CPER.
 - b. The PartitionID needs to be consistent across all RAS API interfaces (for example In-band Management Agent IMA and OOB Management Agent OMA)
4. Header.CreatorID field is a silicon-vendor-assigned GUID that uniquely identifies the type of device that generated this CPER.
 - a. The value is used by the data center to match CPERs produced by a silicon device to the corresponding Analyzer.
 - b. The value must be unique across vendors. Vendors must publish their GUIDs.
 - c. A similar industry example includes PCIe vendor/device IDs or SKU #s.
5. SectionDescriptor.FruID is a GUID derived from a vendor-assigned ID (e.g. serial number) that uniquely identifies the instance of the Field Replaceable Unit (FRU) that had the error.
 - a. This field must be a unique value that allows tracking of error history at the FRU instance level within the fleet.
 - b. In order to create a unique FruID, platform-vendor may use FRU vendor specific information such as x86 PPINs, DIMM serial numbers, etc.
6. SectionDescriptor.FruText field is a platform-assigned human-readable ASCII string that allows data center technicians to quickly identify the faulty Field Replaceable Unit's (FRU) physical location via visual aids in the platform (e.g., silkscreen, label). The value is unique in the platform.

Implementation notes:

In cases where a RAS API endpoint consists of multiple IPs, possibly from different companies, e.g. a CPU core, memory controller, IO interface, etc, the RAS API endpoint must have a single

CreatorID and an associated analyzer for the whole SoC. The routing of CPERs and CPADs inside such an SoC is implementation specific, though IP providers are encouraged to implement standards in CPER and CPAD section bodies to facilitate the implementation of analyzers. Suppliers are encouraged to create RAS API endpoints that cover the installed component, including things like voltage regulators and PCIe retimers.

8.3.2. Common Platform Action Descriptor (CPAD)

This section describes the Common Platform Action Descriptor format for representing platform actions to be performed by RAS API endpoints.

RAS actions are a subset of RAS features designed to initiate specific services, encompassing capabilities and configuration parameters.

8.3.2.1. CPAD General Format

The general format of the CPAD is illustrated below. The fields in CPAD have the same meaning as fields defined in CPER format in UEFI specification unless mentioned otherwise. The record consists of a header; followed by one or more section descriptors; and for each section descriptor, an associated section body which describes an action to be taken. Only one action will be contained in each section descriptor and any associated parameters for the action will be contained in the corresponding section body.

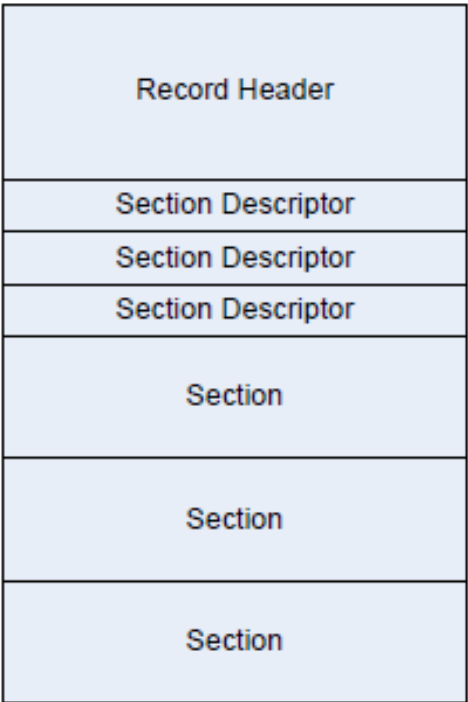


Figure 2 General Format of a Common Platform Action Descriptor (CPAD)

8.3.3. CPAD Record Header

The record header of the CPAD includes information which uniquely identifies a set of actions on a given system. The contents of the record header are described in the [table](#) below.

The 4-byte Severity field in the CPER header definition is replaced with two fields of 1-byte each, namely Urgency and Confidence. The intention of these fields is to indicate to the endpoint if the root cause of the problem is likely to lead to a service interruption. Data center policy defined in this field can indicate predicted time to future service interruption.

The following fields need to be used consistently for routing by fault management infrastructure. These prevent mistakes when end users direct CPADs to the wrong host.

- Platform ID
- Partition ID
- Creator ID

Table 2 – CPAD Record Header

Mnemonic	Byte Offset	Byte Length	Description
Signature Start	0	4	ASCII 4-character array "CPAD" (0x43,0x50,0x41,0x44). Identifies this structure as Common Platform Action Descriptor
Revision	4	2	Consistent with CPER definition (refer to UEFI spec Appendix N)
Signature End	6	4	Consistent with CPER definition (refer to UEFI spec Appendix N)
Section Count	10	2	This field indicates the number (N) of valid CPAD sections associated with the record, corresponding to each of the following section descriptors.

Mnemonic	Byte Offset	Byte Length	Description
Urgency	12	1	The urgency describes if the root cause of the problem is likely to lead to a service interruption: Bit 0 - Urgent (Service disruption expected with maximum probability) 1-7 - Reserved - not defined. Endpoint may honor this field based on vendor policy. The urgency is the highest urgency of the CPAD section descriptors.
Confidence	13	1	Confidence level of the analyzer that is proposing the action. Used by server operators in defining their policies. 0-100% Confidence level The confidence is the highest confidence of the CPAD section descriptors.
Reserved	14	2	Reserved
Validation bits	16	4	This field indicates the validity of the following fields: · Bit 0 – If 1, the PlatformID field contains valid information · Bit 1 – If 1, the TimeStamp field contains valid information · Bit 2 – If 1, the PartitionID field contains valid information · Bits 3-31: Reserved, must be zero.
Record Length	20	4	Indicates the total size of the CPAD in bytes including the size of the record header, all section descriptors, and section bodies. The size may include extra buffer space to allow for the dynamic addition of section descriptors and bodies.

Mnemonic	Byte Offset	Byte Length	Description
Timestamp	24	8	The timestamp correlates to the time when the CPAD was created. Consistent with CPER definition (refer to UEFI spec Appendix N)
Platform ID	32	16	Platform-assigned GUID that uniquely identifies the platform.
Partition ID	48	16	Silicon-vendor -assigned GUID that uniquely identifies the RAS API Endpoint within the platform
Creator ID	64	16	Silicon-vendor assigned GUID that uniquely identifies the entity (e.g., Analyzer) that generated this CPAD
Notification Type	80	16	This field contains a preassigned GUID that associates the action with its type, which could be: • A RAS action for an SoC vendor • A hyperscaler infrastructure serviceability action • A vendor-defined custom RAS action
Record ID	96	8	Unique record ID.
Flags	104	4	Reserved. Set to zero.
Reserved	108	20	Reserved. Set to zero.
Section Descriptor	128	Nx72	Refer to next section

8.3.4. CPAD Section Descriptor

The CPAD Section Descriptor is defined in the table below. The 4-byte Severity field in the CPER Section Descriptor definition is replaced with two fields of 1-bytes each, namely Urgency and Confidence. The intention of these fields is to indicate to the endpoint if the root cause of the problem is likely to lead to a service interruption. Data center policy defined in this field can indicate predicted time to future service interruption.

The following fields need to be consistent for routing by fault management infrastructure. These are required to handle the case where a part was replaced before the CPAD was sent/acted upon:

- FRU ID

Table 3 – CPAD Section Descriptor

Mnemonic	ByteOffset	ByteLength	Description
Section offset	0	4	Offset in bytes of the section body from the base of the record header
Section length	4	4	The length in bytes of the section body
Section revision	8	2	Same definition as CPER.
Validation Bits	10	1	This field indicates the validity of the following fields: · Bit 0 - If 1, the FRUID field contains valid information Bit 1 - If 1, the FRUString field contains valid information Bits 7:2 – Reserved, must be zero.
Reserved	11	1	Reserved
Flags	12	4	Reserved
Section Type	16	16	This field holds a GUID value indicating the format of the section body. The Section Type is independent of the Action ID.
FRU ID	32	16	GUID derived from a vendor-assigned ID (e.g. serial number) that uniquely

Mnemonic	ByteOffset	ByteLength	Description
			identifies the instance of the Field Replaceable Unit (FRU) that had the error
Urgency	48	1	Indicates if the problem is likely to lead to a service interruption (True, False)
Confidence	49	1	Confidence level (0-100%) of the analyzer that is proposing the action.
Reserved	50	2	Reserved
FRU text	52	20	ASCII string identifying the FRU hardwareplatform-assigned human-readable ASCII string that allows data center technicians to quickly identify the faulty Field Replaceable Unit's (FRU) physical location via visual aids in the platform (e.g., silkscreen, label). The value is unique in the platform.
Action ID	72	2	The Action ID identifies the action to be taken. Placing this in the section header allows the CSP to identify the action without parsing the section body. Standard and vendor-proprietary actions may be supported. This field is reserved in version 0.9.

8.3.5. CPAD Section Body

The CPAD section body contains the information needed to perform the action. Vendors may define proprietary section bodies. More details will be provided in the next release of this specification.

9. Glossary of Terms

Key RAS API terminology is listed in Section-5. For a detailed list for Hardware Fault Management terminology please refer to:

[Link to the OCP Hardware Fault Management Glossary](#)

10. References

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.
 2. [OCP Hardware Fault Management Infrastructure Requirements](#)
 3. [OCP Fleet Memory Fault Management Framework Requirements](#)
 4. [UEFI specification - Appendix N](#)
-