Arm® Server Base Manageability Requirements 1.0

Platform Design Document

Non-confidential

Document number: DEN0069B

Server Base Manageability Requirements

Release inormation

The Change History table lists the changes made to this document.

Table 1-1 Change history

Date	Issue	Confidentiality	Change
30 January 2020	A	Non-Confidential	Initial release, SBMR 1.0
15 June 2020	B	Non-Confidential	License LES-PRE-21585

Arm Non-Confidential Document Licence (“Licence”)

This Licence is a legal agreement between you and Arm Limited (“**Arm**”) for the use of the document accompanying this Licence (“**Document**”). Arm is only willing to license the Document to you on condition that you agree to the terms of this Licence. By using or copying the Document you indicate that you agree to be bound by the terms of this Licence. If you do not agree to the terms of this Licence, Arm is unwilling to license this Document to you and you may not use or copy the Document.

“**Subsidiary**” means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries (“Licensee”) is subject to the terms of this Licence between you and Arm.

Subject to the terms and conditions of this Licence, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide licence to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the licence granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the licence granted in (i) above.

Licensee hereby agrees that the licences granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

THE DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENCE, TO THE FULLEST EXTENT PETMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENCE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE’S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENCE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This Licence shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to Licensee. Licensee may terminate this Licence at any time. Upon termination of this Licence by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this Licence, all terms shall survive except for the licence grants.

Any breach of this Licence by a Subsidiary shall entitle Arm to terminate this Licence as if you were the party in breach. Any termination of this Licence shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

If any of the provisions contained in this Licence conflict with any of the provisions of any click-through or signed written agreement with Arm relating to the Document, then the click-through or signed written agreement prevails over and supersedes the conflicting provisions of this Licence. This Licence may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this Licence and any translation, the terms of the English version of this Licence shall prevail.

The validity, construction and performance of this Licence shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: LES-PRE-21585

1 ABOUT THIS DOCUMENT

1.1 Introduction

This document is intended for SBSA[2]-compliant 64-bit Arm based servers. It provides a path to establish a common foundation for server management where common capabilities are standardized and differetiation truly valuable to the end-users are built on top.

This specification leverages the prevalent industry standard system management specifications of Redfish[7], Platform Level Data Model (PLDM)[17] and Management Component Transport Protocol (MCTP)[13]. These specifications are defined in the DMTF Redfish Forum and Platform Management Components Intercommunication (PMCI) Working Group.

1.2 References

This document refers to the following documents.

Reference	Doc No	Authors	Title
[1]	Arm DDI 0487	Arm	Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile
[2]	Arm DEN 0029	Arm	Server Base System Architecture SBSA
[3]	ACPI	UEFI.org	Advanced Configuration and Power Interface Specification
[4]	UEFI Specification	UEFI.org	Unified Extensible Firmware Interface Specificatio
[5]	Arm DEN 0044	Arm	Server Base Boot Requirements SBBR
[6]	DSP0134	DMTF	System Management BIOS (SMBIOS) Reference Specification
[7]	DSP0266	DMTF	Redfish Specification
[8]	DSP0272	DMTF	Redfish Interoperability Profiles Specification
[9]	DSP0270	DMTF	Redfish Host Interface Specification
[10]	DSP8010	DMTF	Redfish Schema
[11]	DSP8011	DMTF	Redfish Standard Registries
[12]	DSP8013	DMTF	Redfish Interoperability Profiles Bundle
[13]	DSP0236	DMTF	MCTP Base Specification
[14]	DSP0237	DMTF	MCTP SMBus/I2C Transport Binding Specification
[15]	DSP0238	DMTF	MCTP PCIe VDM Transport Binding Specification
[16]	DSP0239	DMTF	MCTP IDs and Codes
[17]	DSP0240	DMTF	PLDM Base Specification
[18]	DSP0241	DMTF	PLDM Over MCTP Binding Specification

[19]	DSP0245	DMTF	PLDM IDs and Codes Specification
[20]	DSP0248	DMTF	PLDM for Platform Monitoring and Control Specification
[21]	DSP0222	DMTF	NC-SI Specification
[22]	DSP0261	DMTF	NC-SI over MCTP Binding Specification
[23]	DSP0267	DMTF	PLDM for Firmware Update Specification
[24]	DSP0218	DMTF	PLDM for Redfish Device Enablement Specification
[25]	DSP0235	DMTF	NVMe over MCTP Binding Specification
[26]	IPMI	Dell, HP, Intel, NEC	Intelligent Platform Management Interface 2.0
[27]	Arm DEN 0022	Arm	Power State Coordination Interface
[28]	Arm DEN 0056	Arm	System Control and Management Interface
[29]	OCP Baseline Redfish Profile	OCP	OCP Baseline Hardware Management Redfish Profile
[30]	OCP Server Redfish Profile	OCP	OCP Server Hardware Management Redfish Profile
[31]	NIST800-147	NIST	BIOS Protection Guidelines
[32]	NIST800-193	NIST	Platform Firmware Resiliency Guidelines
[33]	NIST800-155	NIST	BIOS Integrity Measurement Guidelines
[34]	https://www.opencompute.org/wiki/Server/SpecsAndDesigns	OCP	Open Server Specs and Designs

[35]	DSP0249	DMTF	Platform Level Data Model (PLDM) State Set Specification
[36]	Arm IHI 0031	Arm	Arm® Debug Interface Architecture Specification, ADIv5
[37]	Arm IHI 0074	Arm	Arm® Debug Interface Architecture Specification, ADIv6
[38]	DSP0256	DMTF	MCTP Host Interface Specification
[39]	SBSG	Arm	Server Base Security Guide (SBSG)
[40]	http://openocd.org/doc/pdf/openocd.pdf	OpenOCD	Open OCDUser Guide

1.2.1 Cross References

This document cross-references sources that are listed in the References section by using the section sign §.

Examples:

ACPI § 5.6.5 - Reference to the ACPI specification [3] section 5.6.6

UEFI § 6.1 - Reference to the UEFI specification [4] section 6.1

1.3 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
ACPI	Advanced Configuration and Power Interface.
BMC	Baseboard Management Controller
Host	The Computer System that is managed
Host Software	The software running on the Host, including Operating System and its Software components (such as drivers or applications), as well as preboot software such as UEFI drivers and applications
IPMI	Intelligent Platform Management Interface. It defines common interfaces that allow IT managers to receive status alerts, send instructions to servers and run diagnostics over a network versus locally at the server.
MCTP	Management Component Transport Protocol. A transport independent protocol that is used for intercommunication within an MCTP Network (consists of one or more physical transports that are used to transfer MCTP Packets between MCTP Endpoints.

NC-SI	Network Controller Sideband Interface. The interface (protocol, messages, and medium) between a Management Controller and one or more Network Controllers. It is responsible for providing external network connectivity for the Management Controller while also allowing the external network interface to be shared with traffic to and from the host
OEM	Original Equipment Manufacturer. In this document, the final device manufacturer.
PLDM	Platform Level Data Model. An internal facing low level data model that is designed to be an effective data/control source for mapping under the Common Information Model (CIM). It defines data structures and commands that abstract platform management subsystem components.
Redfish Interface	An open industry standard specification that specifies a RESTful interface and schema for hardware management, and that allows users to integrate solutions within their existing tool chains. Extensions to Redfish can also be made. Swordfish for example is a SNIA standard that builds upon Redfish's local storage management capabilities to address enterprise storage devices
SiP	Silicon Partner. In this document, the silicon manufacturer.
UEFI	Unified Extensible Firmware Interface.
UEFI Boot Services	Functionality that is provided to UEFI Loaded Images during the UEFI boot process.
UEFI Runtime Services	Functionality that is provided to an Operating System after the ExitBootServices() call.
Satellite Management Controller (SatMC)	A microcontroller or processor that interpret and process management-related data, and initiate management-related actions on management devices. It may be part of SOC or can be outside of SOC.
Baseboard management controller (BMC)	The main management controller in an standards-based, remotely managed platform management subsystem. Also sometimes used as a generic name for a motherboard-resident management controller that provides motherboard-specific hardware monitoring and control functions for the platform management subsystem.

1.4 Feedback

Arm welcomes feedback on its documentation.

If you have comments on the content of this manual, send an e-mail to errata@arm.com. Give:

• The title (Server Base Manageability Guide).

• The document ID and version (DEN0069 1.0).

• The page numbers to which your comments apply.

• A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

2 SCOPE AND BACKGROUND

This document provides a path to establish a common foundation for server management on SBSA-compliant Arm AArch64 servers where common capabilities are standardized and differetiation truly valuable to the end-users are built on top.

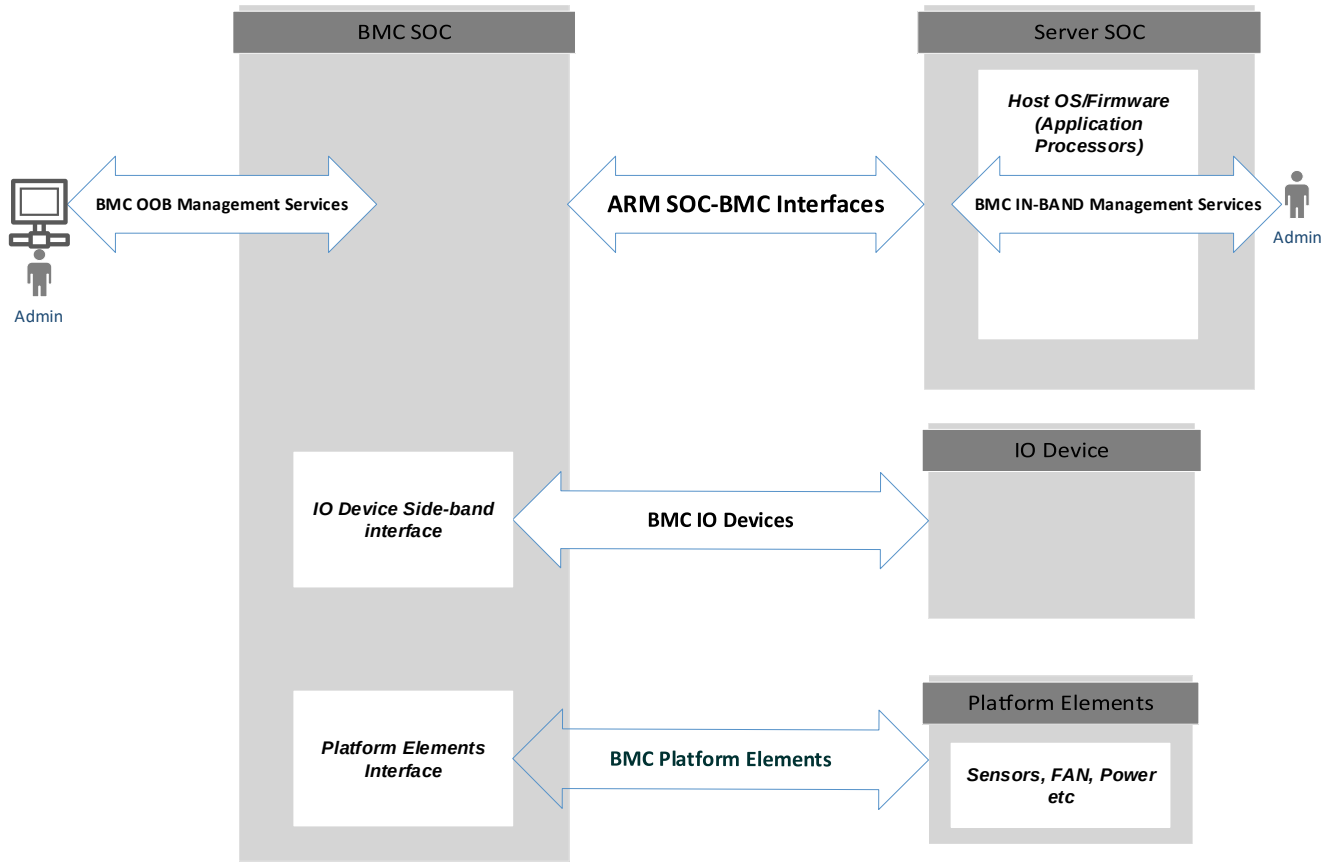
2.1 Scope

Redfish [7], PLDM [17], and MCTP [13] specifications have been chosen to ease the adoption of Arm, by aligning the AArch64 server ecosystem to where the existing enterprise server market is moving to.

Redfish is based on industry standard RESTful interface for IT infrastructure. Redfish uses the secure or standard Hypertext Transfer Protocol (HTTP/HTTPS) to transport resources and configure operations. Resources (in payload) are JavaScript Object Notation (JSON) formatted, making them eaqually usable by apps, GUIs and scripts. Redfish resources are schema-backed and human readable, with schemas [10] defined using JSON Schema, OData 4.0, or OpenAPI formats. Redfish provides a secure, multi-node capable replacement for IPMI-over-LAN [26]. It is intended to meet Open Compute Project (OCP) [29][30] remote machine management requirements.

The support for the legacy Intelligent Platform Management Interface (IPMI)[26] is still required as IPMI-based tools are still widely used by the endusers. The IPMI contributors group is no longer accepting requests for contribution. There is no venue for Arm and its ecosystem partners to change or improve the specification. The adoption of IPMI is therefore “as is”. As the industry becomes ready, this document may make the IPMI support optional.

Figure 3-1 - Server Management Interfaces



This document addresses the need to establish the following common standard interface sets (See Figure 3-1):

1. Arm SoC-BMC (Baseboard Management Controller) Interfaces: used by the BMC and SoC to communicate with each other. Some examples are described in section 2.2.
2. BMC-Platform Elements Interface: used by the BMC to communicate with the Platform Elements (e.g., devices, sensors)
3. BMC-IO Device Interface: used by the BMC to communicate with one type of the Platform Elements: the IO devices
4. BMC Management Services (Out-of-Band) Interface: used by the System Admins via external network to manage servers remotely

The focus of this document is to provide manageability requirements for various SBMR compliance levels (i.e. Level Mx as described in the next section). These are requirements with respect to relevant interfaces between the Arm SoC and the BMC, as described in the Table 1 summary below.

This document may also provide some guidance and recommendations with respect to other BMC interfaces with IO devices and platform elements.

2.2 Background

Typically, there are several interfaces used for communication and interaction between the Arm SoC and the BMC.

Host/SoC In-band Interface

This interface is used by the Host Software (i.e. OS/Hypervisor/User Software) and/or System Firmware (e.g. UEFI [4]) to communicate with the BMC . It is typically exposed to Host Software via SMBIOS [6], ACPI [3] tables (e.g. SPMI), and/or PCIe configuration space. Earlier Arm server systems are IPMI based, with newer Arm server systems transitioning to Redfish [7] and MCTP host interface [38].

Typical use cases of this interface include:

- UEFI – BMC communication (via IPMI on earlier Arm server systems, and via MCTP host interface on later Arm server systems):
  - Reporting SMBIOS [6] table
  - Reporting boot progress codes
  - Error reporting
  - General event logging

- OS/Hypervisor software – BMC communication
  - Redfish Authentication (via IPMI on earlier Arm server systems, possibly MCTP host interface [38] on later Arm server systems)

- User software – BMC communication
  - User/Admin access to BMC management services (via IPMI and/or Redfish [7]), for local server configuration, update, deployment, or monitoring.

SoC Side-band Interface

This interface is used by the BMC firmware to communicate with the SoC via a “Satellite Management Controller” (SatMC). Typical use-cases may include:

- Early stages of boot progress codes reporting
- Telemetry (Temperature, power etc.)
- RAS error reporting
- Early stages of boot event logging

PCIe connection between the Arm SoC and the BMC

This interface may exist for the following use cases:

- Remote KVM session using PCIe for exposing a graphics controller (typically implemented in the BMC) for the host's video output
- MCTP side-band communication between the BMC and PCIe devices via PCIe VDM path (Note: in this usage, the Arm SoC must contain the logic to route the PCIe VDM messages to the proper IO devices)
- Shared memory mailbox communication between the BMC and the SoC host software

USB connection between the Arm SoC and the BMC

This interface may exist for the following use cases:

- Remote Media session using USB for exposing a virtual media (CD-ROM, Floppy, Memory stick)
- Remote KVM session using USB for exposing Keyboard/Mouse devices
- Redfish Host Interface using USB for exposing a Network-over-USB interface

NOTE: This interface may not be directly connected / integrated in the Arm SoC. It could be an external onboard PCIe-based USB controller or PHY that connects to the BMC USB ports.

JTAG connection between the Arm SoC and the BMC

This interface may exist for the following use-cases :

- Remote hardware debug (e.g. breakpoints, single stepping, etc.) using JTAG interface and exposed over BMC management network
- Crash dump or scan dump feature (for crash or hang scenarios) using JTAG interface and exposed over BMC management network
- Memory/Register dump features using JTAG interface and exposed over BMC management network

NOTE: Debug security must be considered on production platforms, either permanently disabled or re-enabled through authentication per IMPLEMENTATION DEFINED mechanisms.

Additional connectivity (various physical media) between the Arm SoC and the BMC

Such interfaces may exist for the following use cases:

- Access to the Arm SoC thermal and power information and control
- Access to the Arm SoC RAS error information and control

2.3 Arm SoC-BMC Interface Terminology

This document will use a specific terminology and definition to refer to different types. For example, terms like "In-Band", "Side-Band", and "Out-Of-Band" have a specific meaning when discussing interfaces to/from the BMC. These terms relevant to the areas covered are defined in this section.

Table 2-1 Arm SoC-BMC Interface Terminology

Name	Master	Slave	Description / Example / Notes	In SBMR Scope?
SoC **In-band** Interface	Arm SoC (Host OS / FW)	BMC	This is typically IPMI SSIF (I2C interface) or Redfish Host Interface (via USB/PCIe) or MCTP Host Interface (MCTP over KCS, MCTP over Serial) NOTE**:** This interface is invasive to the main processor complex (i.e. processing cycles is required).	Yes
SoC **Side-band** Interface	BMC	SoC / SatMC	Replaces SMLINK on x86. This interface may leverage a proprietary protocol or a more standard MCTP transport protocol (physical interface specifics depend on the system implementors). **This is a "multi-master" bi-directional communication interface.** NOTE**:** This could be a SatMC within the SoC or an intermediary entity.	Yes
Out-of-Band Interface	Datacenter management network	BMC	This is typically IPMI or Redfish commands via management network	Yes
SoC **Debug** Interface (i.e. JTAG)	BMC	SoC	This is the JTAG debug interface used for hardware debugging the software and possibly firmware executing on the	Yes
BMC notification pins (e.g. GPIOs or dedicated pins)	SoC	BMC	These pins are used for high priority notifications from the SoC to the BMC (e.g. critical thermal events or SoC errors). NOTE: Some pins may be bi-directional (e.g. PROCHOT)	Partially Covered
SoC notification pins (e.g. GPIOs or dedicated pins)	BMC	SoC	These pins are used for high priority notifications from the BMC to the SoC (e.g. critical thermal events or SoC errors). NOTE: Some pins may be bi-directional (e.g. PROCHOT)	Partially Covered
Serial Console (i.e. UART)	SoC	BMC	Used for implementing Serial-over-LAN (SoL). Arm SoC typically have at least one or more UARTs.	Yes

			Must be an Arm SBSA [2] compliant UART controller on the SoC side. Default Baud rate for interoperability with commercially available BMCs is required to be 115200 bits/second.	
IO Device Side-Band Interfaces (Broad range of various interfaces)	BMC	IO Devices (attached to the Arm SoC)	This is referring to IO devices attached to the Arm SoC that the BMC may need to monitor and/or manage. Examples of such IO devices may include side-band interface to firmware storage device (e.g. UEFI SPI-NOR flash) and PCIe cards. NOTE: These interfaces are only partially in scope of the SBMR compliance requirements. Some recommendations and guidance may be provided based on external specifications and standards. This specification will not cover the security aspects of these side-band interfaces (e.g. platform root-of-trust (ROT) chips which manage and authenticate traffic on these side-band interfaces).	Partially Covered
N/A (Broad range of various Interfaces)	BMC	Platform Elements	This may include a broad range of interfaces (e.g. power supplies, voltage regulators, platform sensors, etc.) NOTE: This interface is outside the scope of the SBMR compliance requirements. Some recommendations and guidance may be provided based on external specifications and standards.	No

3 COMPLIANCE LEVELS AND REQUIREMENTS

This document defines a number of levels of manageability compliance (e.g., M0, M1, M2) with the intention of steering the partners to gradually move to the Redfish and PLDM / MCTP standard environment. There is no direct linkage between these levels and the SBSA levels.

This specification defines a set of requirements for each compliance level. The compliance levels include M1, M2, M3a, and M4a.

NOTE: *M3a and M4a describes preliminary definitions of future compliance levels (for the purpose of public review and feedback). The 'α' denotes that this is an "alpha" (i.e. unofficial) compliance level. These definitions are subject to change in future publications of this specification.*

The table below shows the summary of SBMR compliance levels.

Table 3-1 SBMR Complaicne Levels

Level	Out-of-band Interface	SoC Side-band Interface	Host/SoC In-band Interface	BMC-IO Device Interface	BMC Platform Element Interface
M0	IMPDEF	IMPDEF	IMPDEF	IMPDEF	IMPDEF
M1	**Required**: IPMI	IMPDEF	**Required:** IPMI SSIF	IMPDEF	IMPDEF
M2	**Required**: Redfish and IPMI	IMPDEF	**Required**: IPMI SSIF and Redfish Host Interface	**Conditional Requirement:** If shared physical NIC interface - NC-SI is required	IMPDEF
M3α	**Required**: Redfish	**Required:** MCTP over I2C/SMBUS	**Required:** Redfish Host Interface **and** IPMI SSIF **or** MCTP (Physical Interface TBD)	**Conditional Requirement:** If shared physical NIC is used: NC-SI over RBT or MCTP	Refer to [34] and [26] for guidance
M4α	**Required**: Redfish	**Required:** MCTP over I3C **or** MCTP (Physical Interface TBD)	**Required:** Redfish Host Interface **And** MCTP Host Interface (Physical Interface TBD)	**Required:** NVMe over MCTP **Conditional Requirement:** If shared physical NIC is used: NC-SI over RBT or MCTP	**Conditional Requirement:** Redfish/PLDM /MCTP

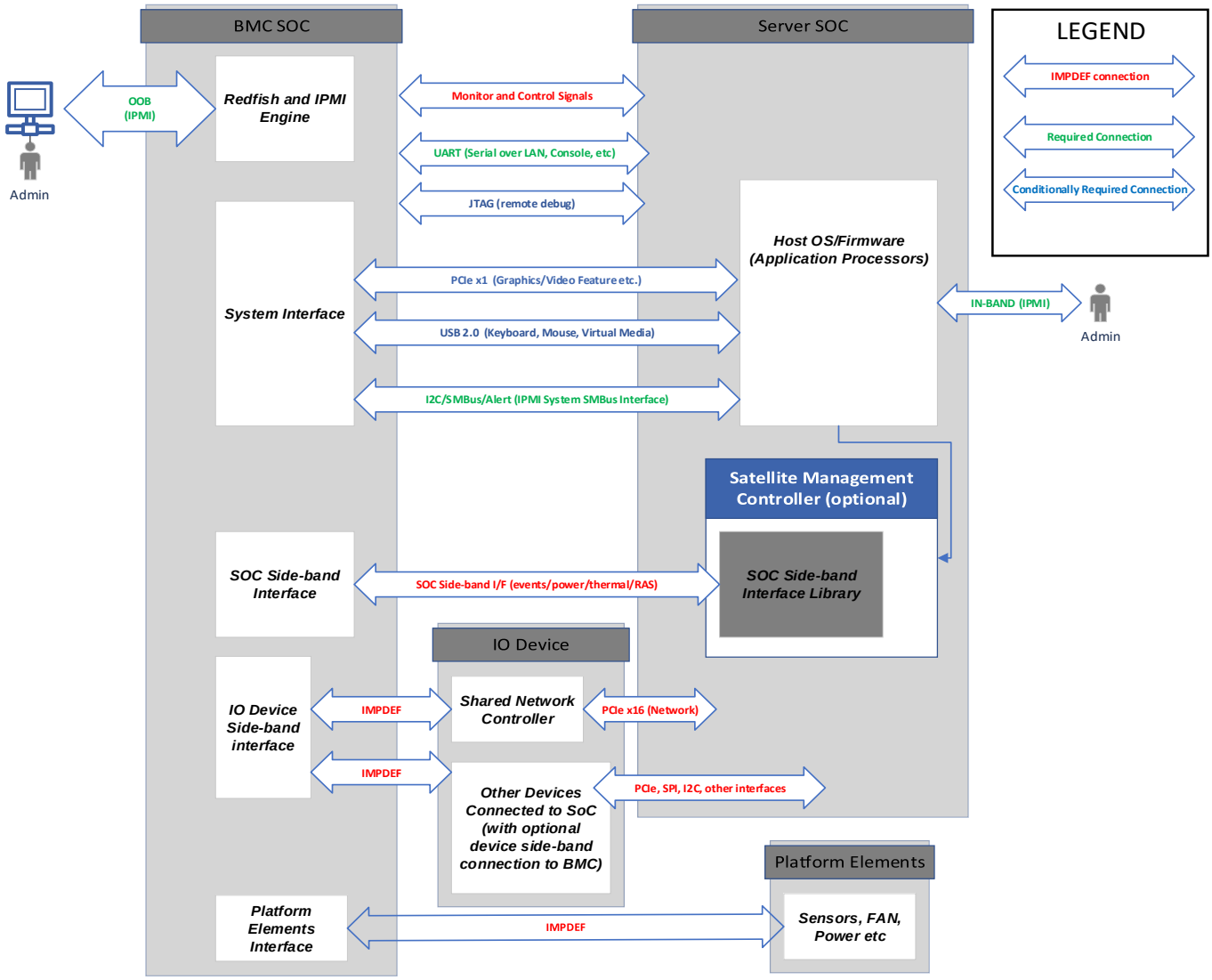
3.1 Level M0

Server management for the Level M0 based server systems are IMPLEMENTATION DEFINED.

There is no standardization for the server management interfaces. Typically, some variations of IPMI based implementations are used to provide the interfaces from the SoC-BMC Interfaces, Host Interface, BMC-Platform Elements Interface, BMC-IO Device Interface and BMC Management Services Interface.

3.2 Level M1

Figure 3-2 Server Management Interfaces (Level M1)



3.2.1 SoC-BMC Interfaces

Most SoC-BMC interfaces for the Level M1 based server systems are IMPLEMENTATION DEFINED. with the exceptions of the requirements described in the following subsection.

3.2.1.1 Requirements

3.2.1.1.1 Host SoC In-Band Interface

M1 compliance requires that an IPMI interface must be supported for communication from the Arm SoC to the BMC. The IPMI specification [26] defines four supported physical/logical interfaces (KCS, BT, SMIC, and SSIF). SBMR requires IPMI SSIF as the preferred interface for IPMI in-band communication.

The Arm SoC must have a **SMBus System Interface (SSIF)** connection to the BMC for IPMI communication as described by the IPMI specification. At minimum, this must be an I2C connection used for sending IPMI commands to the BMC. It is recommended that an “ALERT” pin is also supported to enable BMC notification to the host.

3.2.1.1.2 Console UART

The Arm SoC must have at least one SBSA [2] compliant UART connection to the BMC for the purpose of serial-over-LAN (SoL) support (e.g. for OS / UEFI console purposes, output/input).

Per the SBSA [2] and SBBR [5], the console UART must be an SBSA [2] compliant UART that must be exposed to the host software via the Serial Port Console Redirection Table (SPCR).

Additional UART console connections from the Arm SoC to the BMC are permitted but are considered IMPLEMENTATION DEFINED.

Default baud rate for interoperability with commercially available BMCs is required to be 115200 bits/second.

3.2.1.2 Recommendations

3.2.1.2.1 PCIe

If remote Keyboard-Video-Mouse (KVM) is supported on the platform, it is strongly recommended that the Arm SoC have a PCIe connection to the BMC for the purpose of graphics (e.g. VGA).

3.2.1.2.2 USB

If remote Virtual Media or KVM is supported on the platform, it is strongly recommended the Arm SoC have a USB host connection (either via on-chip/SoC USB controller or external onboard USB controller) to the BMC for the purpose of enabling remote keyboard, mouse, and virtual media.

3.2.1.2.3 JTAG

Remote Debug is an invasive or non-invasive external debug, through a physical interface (i.e. JTAG), that is remotely controlled through an Out-of-band interface exposed by the platform BMC. Examples of Remote Debug functions include:

- Crash dump analysis
- Register and memory inspection.
- Stepping through code.
- Low-level bare metal analysis.

If support for JTAG based remote debug and crash dump functions is needed, an IEEE 1149.1 JTAG interface is required:

- Control of the JTAG interface can be exposed over the Out-of-band interface.
- Inclusion of control of the TRST signal on the BMC is required.

- Inclusion of the TRST signal on the SoC is IMPLEMENTATION DEFINED.
- Where multiple SoCs which need support for remote debug functions are connected to the same BMC, the JTAG interfaces shall be daisy-chained, for control by a single JTAG interface on the BMC.

Access to some or all debug functionality might be prevented at certain lifecycle states of the SoC. When such access is prevented, an IMPLEMENTATION DEFINED mechanism should be provided to enable Remote Debug access.

NOTE: *For more guidance on debug and JTAG security, refer to the Arm Server Base Security Guide (SBSG) [39]*

Where a JTAG interface is provided for Remote Debug functions and when Remote Debug access is enabled, the JTAG interface shall provide access to all TAP controllers that are compliant with the Arm Debug Interface, ADIv5 [36] or ADIv6 [37].

- The Arm Debug Interface TAP controllers shall provide access to the following for each Arm processor that needs Remote Debug access:
  - The external debug interface.
  - The external debug interface for any Cross-Trigger Interfaces (CTI).
  - The external debug interface for any Performance Monitor Units (PMU).
  - The external debug interface for any processor trace functions (e.g. ETM).
- The Arm Debug Interface TAP controllers shall provide access to all components required to route trace from the processor trace source to any trace sinks.
- Access to other debug functionality is IMPLEMENTATION DEFINED.
- The Arm Debug Interface TAP controllers shall provide access to all components required to enable access to any of the above components, for example ROM tables and power control requests.

For more details, refer to Appendix E .

3.2.2 BMC-Platform Elements Interface Recommendations

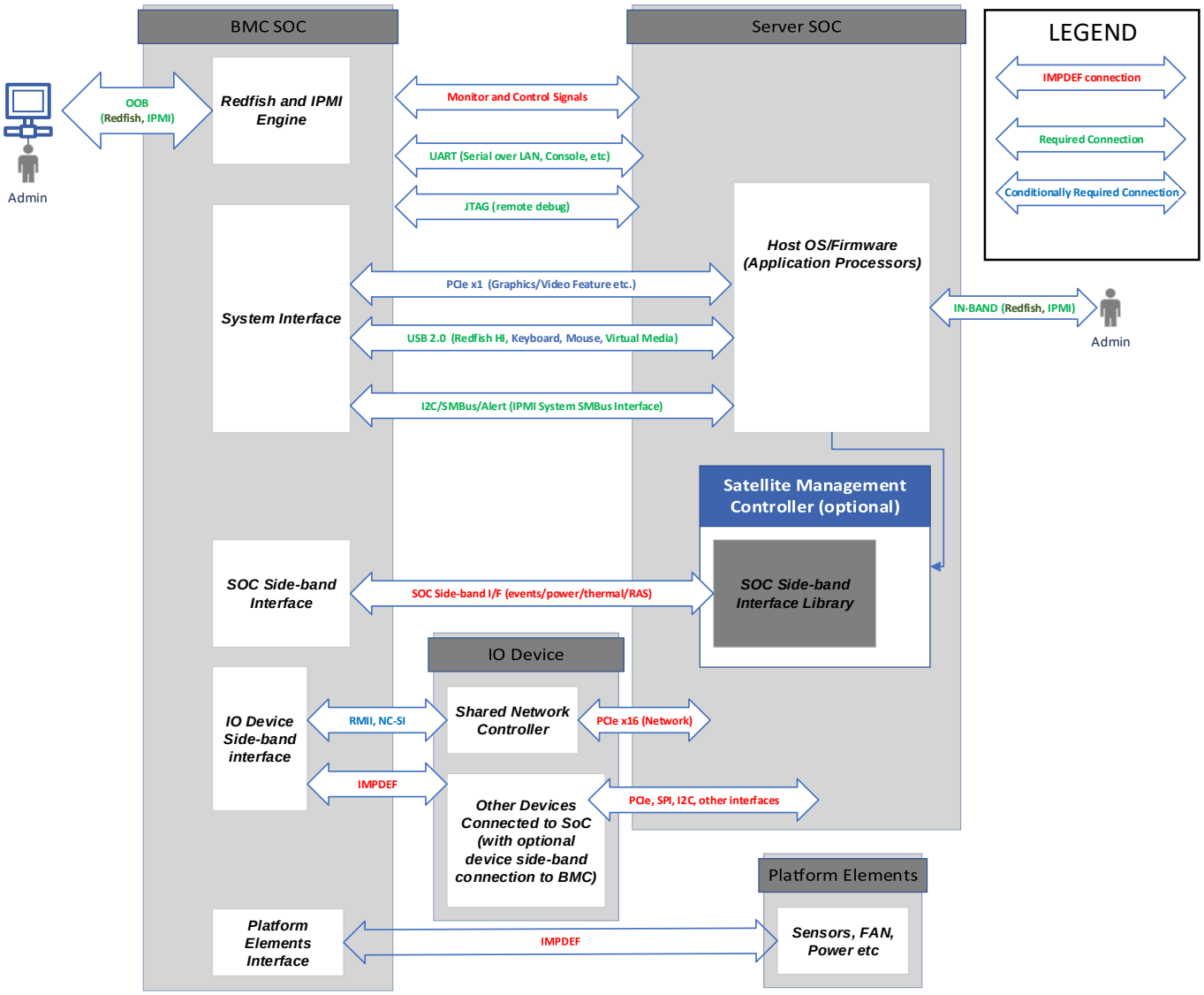
The BMC-Platform Elements interface for the Level M1 based server systems is IMPLEMENTATION DEFINED. Typically, the SMBus/I_2C medium is used.

3.2.3 BMC Management Services (Out-of-Band) Interface Recommendations

Support for IPMI is a requirement for M1 compliant server systems. Refer to Appendix B for minimal IPMI commands required.

3.3 Level M2

Figure 3-3 Server Management Interfaces (Level M2)



3.3.1 SoC-BMC Interfaces

3.3.1.1 **Requirements**

The requirements for these interfaces on the Level M2 based server systems are the same as those of the Level M1 based server systems, with some additional requirements.

3.3.1.1.1 Host SoC In-Band Interface

The Host/SoC In-Band interface must be compliant to the Redfish Host Interface Specification [9]. The Arm SoC must expose this interface via compliant physical connection, namely PCIe or USB based. One of the following physical interface requirements must be met:

1) The Arm SoC must have a USB connection (either via on-chip USB support or external onboard USB support via PCIe USB device) to the BMC for Redfish communication over USB network device. At a minimum, this must be USB 2.0 connection or faster.

   Or

2) The Arm SoC must have a PCIe connection to the BMC for Redfish communication over PCIe network device.

NOTE: In addition to USB and PCIe network device, the Redfish Host Interface Specification [9] defines an OEM proprietary method. This proprietary method is not recommended for M2 compliant systems.

In addition, M2 compliance requires that a second Host/SoC In-Band interface based on IPMI that must exist.

3.3.1.1.2 JTAG

JTAG connection between the BMC and the SoC is upgraded from a conditional requirement in Level M1 to a mandatory requirement in Level M2 based server systems.

3.3.1.2 Recommendations

The recommendation for the BMC-SOC interfaces on the Level M2 based server systems are the same as those of the Level M1 based server systems.

3.3.2 BMC-Platform Elements Interface Recommendations

The BMC-Platform Elements interface for the Level M2 based server systems is IMPLEMENTATION DEFINED.

3.3.3 BMC-IO Device Interface Recommendations

If using shared physical NIC interface between BMC and SOC, then Network Controller Sideband Interface (NC-SI)[21] over reduced media independent interface (RMII) based transport is required for Level M2 based server systems.

Network Controller Sideband Interface (NC-SI)[21] defines a combination of logical and physical paths that interconnect the BMC and Network Controller(s) for the purpose of transferring management communication traffic among them. NC-SI includes commands and associated responses, which the BMC uses to control the status and operation of the Network Controller(s). NC-SI also includes a mechanism for transporting management traffic and asynchronous notifications.

The BMC-IO Device Interface for all other IO devices for the Level M2 based server systems is IMPLEMENTATION DEFINED.

3.3.4 BMC Management Services (Out-of-Band) Interface Recommendations

Level M2 based server systems requires that the BMC Management Services Interface supports the Redfish Interface [7].

In addition, IPMI support is also a requirement for M2 compliant server systems. Refer to Appendix B for minimal IPMI commands required.

Level M2 based server systems further standardize the BMC Management Services Interface by adopting the Redfish Interoperability Profiles Specification[8] and the individual profiles contained in the Redfish Interoperability Profiles Bundle[12].

Supporting OCP profiles is required for OCP servers. OCP currently defines two Redfish profiles for hardware management:

1. OCP Baseline Hardware Management Redfish Profile [29]. This is the minimum level a Redfish interface must provide for OCP compliant hardware management.
2. OCP Server Hardware Management Redfish Profile [30]. This profile defines additional requirements on top of the OCP Baseline profile [29] for OCP compliant server hardware management.

As Redfish Schema [10] definitions are designed to provide significant flexibility and allow conforming implementations on a wide variety of products, very few properties within the Schemas are required by the Redfish specification. However, consumers and software developers need a more rigidly defined set of required properties (features) in order to accomplish management tasks. This set allows users to compare implementations, specify needs to vendors, and allows software to rely on the availability of data. To provide that "common ground", a Redfish Interoperability Profile allows the definition of a set of schemas and property requirements, which meet the needs of a particular class of product or service.

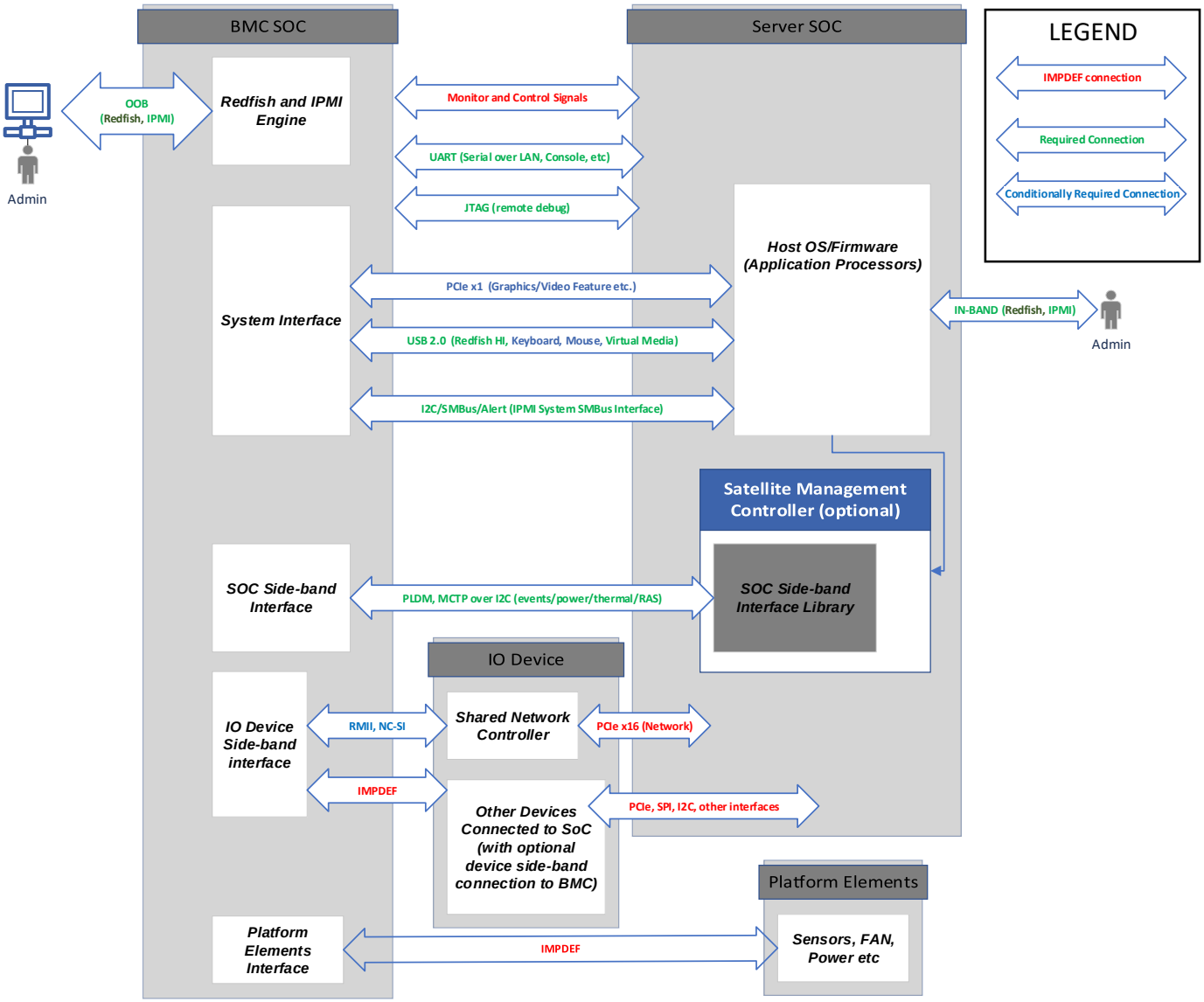
A tool to verify the compliance of a Redfish implementation to the required Redfish profile is available from DMTF at: https://github.com/DMTF/Redfish-Interop-Validator.

NOTE: Arm has the ability to publish Arm-specific profiles if needed, but the intent is to adopt the standard profiles (e.g., OCP profile[29][30]).

3.4 Level M3 alpha (Work-in-progress)

The following block diagram describes the BMC interfaces

Figure 3-4 Server Management Interfaces (Levem M3a)



3.4.1 Requirements

The requirements for these interfaces on the Level M3 based server systems are the same as those of the Level M2 based server systems, with some additional requirements.

3.4.2 SoC-BMC Interface

Level M3 based server systems standardize this interface based on the DMTF PMCI workgroup standards which define specifications for primary intercommunication interfaces/data models between baseboard management controller (BMC) and satellite management controller (SatMC).

- PLDM [17][19][20][23] for the purpose of supporting platform-level data models and platform functions. PLDM is designed to be an effective interface and data model that provides efficient access to low-level platform inventory, monitoring, control, event, and data/parameters transfer functions. PLDM defines data representations and commands that abstract the platform management hardware.

- MCTP [13][16] as a transport protocol format that is independent of the underlying physical bus properties, as well as the "data-link" layer messaging used on the bus.

- PLDM over MCTP binding [18] as the format of PLDM over MCTP messages.

For Level M3 based server systems, the physical and data-link layer methods for MCTP communication are defined by the MCTP over SMBus/I2C binding specification [14].

3.4.3 BMC-Platform Elements Interface Recommendations

For recommendations/guidance on the BMC-Platform Elements interface for the Level M3 based server systems, please refer to Intelligent Platform Management Interface v2.0 (IPMI) specification [26].

For a list of IPMI commands which aid in monitoring and control of platform elements refer to Appendix D .

3.4.4 BMC-IO Device Interface Recommendations

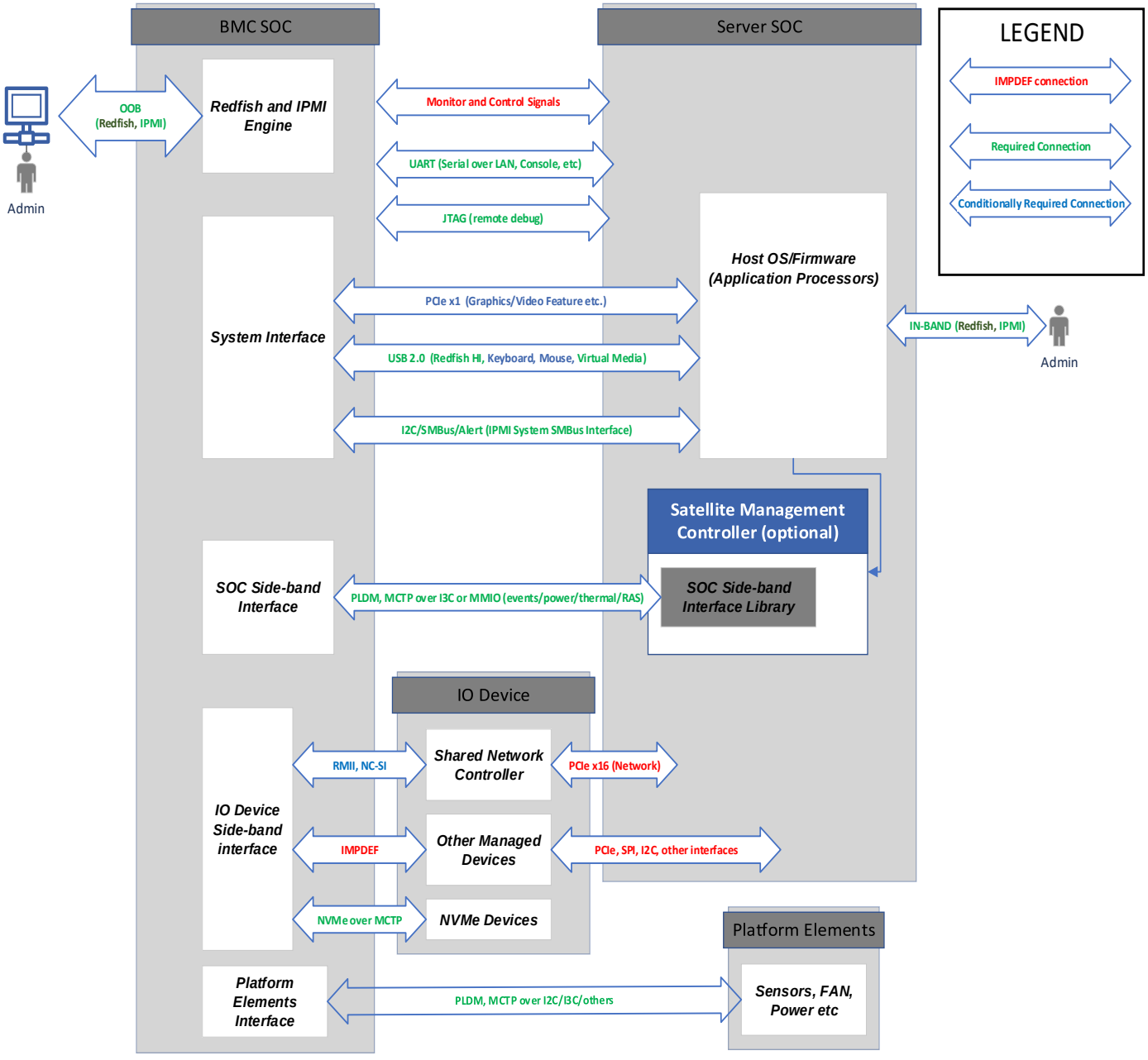
If using shared physical NIC interface between BMC and SOC, then Network Controller Sideband Interface (NC-SI)[21] over reduced media independent interface (RMII) based transport or MCTP is required for Level M3 based server systems. If NC-SI over MCTP [22] is used, the physical layer used is one of the transport bindings on which MCTP can be bound (for example, PCIe VDM or SMBus/I2C).

The BMC-IO Device Interface for all other IO devices for the Level M3 based server systems is IMPLEMENTATION DEFINED.

3.5 Level M4 alpha (Work-in-progress)

The following block diagram describes the BMC interfaces

Figure 3-5 Server Management Interfaces (Level M4a)



3.5.1 Requirements

The requirements for these interfaces on the Level 4 based server systems are the same as those of the Level M3 based server systems, with some additional requirements.

3.5.2 SoC-BMC Interface

For Level M4 based server systems, the physical and data-link layer methods for MCTP communication will be defined by the MCTP over I3C binding specification. Further, high speed memory mapped interface may also be considered. (TBD)

3.5.3 BMC-Platform Elements Interface Recommendations

Level M4 based server systems standardize this interface based on the DMTF PMCI workgroup standards which define specifications for primary intercommunication interfaces/data models between Management Controller (BMC) and managed entities (Platform Elements).

- PLDM[17][19][20][23] for the purpose of supporting platform-level data models and platform functions. PLDM is designed to be an effective interface and data model that provides efficient access to low-level platform inventory, monitoring, control, event, and data/parameters transfer functions. For example, temperature, voltage, or fan sensors can have a PLDM representation that can be used to monitor and control the platform using a set of PLDM messages. PLDM defines data representations and commands that abstract the platform management hardware.
- MCTP [13][16] as a transport protocol format that is independent of the underlying physical bus properties, as well as the "data-link" layer messaging used on the bus.
- PLDM over MCTP binding [18] as the format of PLDM over MCTP messages.
- PLDM for Redfish Device Enablement [24] as the messages and data structures used for enabling PLDM devices to participate in Redfish-based management.

This approach abstracts the potential evolutions of the underlying physical medium, enabling future transport bindings to be defined to support additional media without affecting the base MCTP specification. For the current popular SMBus/I_2C medium, the physical and data-link layer methods for MCTP communication are defined by the MCTP over SMBus/I_2C binding specification [14].

For a list of PLDM commands which aid in monitoring and control of platform elements refer to Appendix D.

3.5.4 BMC-IO Device Interface Recommendations

If using shared physical NIC interface between BMC and SOC, the requirements for these interfaces on the Level 4 based server systems are the same as those of the Level M3 based server systems

Further, Level M4 based server systems standardize NVMe Management Interface support with NVMe Management Messages over MCTP.

Non-Volatile Memory Express (NVMe-MI) is an optimized register interface, command set, and feature set for PCIe based storage. The NVMe Management Interface protocol may also be used for other types of non-volatile memory devices. NVMe Management Interface Commands are used for the accessing configuration, control, and status functions in NVMe-compatible non-volatile memory devices. NVMe Management Messages over MCTP Specification [25] defines how NVMe Management Interface Commands are encapsulated in MCTP Messages and transferred between MCTP Endpoints over the specified transports (for example, PCIe VDM or SMBus/I2C).

The BMC-IO Device Interface for all other IO devices for the Level M4 based server systems is IMPLEMENTATION DEFINED.

APPENDIX A OPENBMC

The OpenBMC project (https://www.openbmc.org/) can be described as a Linux distribution for an embedded device that serves as a BMC for typically, but not limited to, things like servers, top of rack switches or RAID appliances. The OpenBMC stack uses technologies such as Yocto, Open-Embedded, Systemd and Dbus to allow easy customization for each server platform.

OpenBMC is a a Linux Foundation project hosted at https://github.com/openbmc/openbmc. Facebook, Google, IBM, Intel, and Microsoft are the founding TSC members. Arm is now a TSC member.

OpenBMC is a sample implementation of the BMC software. Actual deployment of BMC in SBSA[2] compliant AArch64 servers can chose to use this implementation or other commercial solutions.

APPENDIX B IPMI IMPLEMENTATION GUIDE

This appendix documents the minimum IPMI Commands required.

B.1 Remote Power Control

B.1.1 Power On

A platform must provide a mechanism for remotely powering an individual node on and initiating the boot sequence.

B.1.2 Power Off

A platform must provide a mechanism for remotely powering an individual node off. This mechanism should be provided out-of-band of and without dependencies on the host operating system. For example, graceful power off facilities which rely on the host OS to perform the shutdown would not be sufficient.

B.1.3 Graceful Power Off

A platform must provide a mechanism for remotely initiating an OS-controlled power down of a system.

B.1.4 IPMI Commands Required

IPMI LAN Chassis Power Commands

B.2 Boot Device Selection

Platforms must provide a mechanism to remotely select either a local boot or a network boot on the next system power up.

B.2.1 IPMI Commands Required

IPMI LAN chassis boot device command

B.3 BMC / Host Mapping

It should be possible to automatically determine the mapping between a host and its BMC. Either the host must be able to identify its BMC configuration through an in-band mechanism, or the BMC must be able to provide unique identification information about the host (e.g. Host MAC addresses).

B.4 BMC User Manipulation

When an IPMI LAN capable BMC is used to provide platform interfaces, it must be possible for the deployment server to authenticate to the BMC by using the IPMI System Interface to add a private user to the BMC via the host operating system. The System Interface does not require the user to authenticate to the BMC to manipulate the user settings. Once the deployment server has defined a user on the BMC, the MAAS server will be able to authenticate to the BMC over the IPMI LAN interface. This requires an IPMI-compliant BMC system Interface.

B.5 IPMI Support Verification

A script to verify the basic remote IPMI functionality is available here:

https://git.launchpad.net/~ce-hyperscale/maas/+git/maas/plain/maas-ipmi-test.sh?h=maas-bmc-tests

APPENDIX C RAS MESSAGE FORMATS

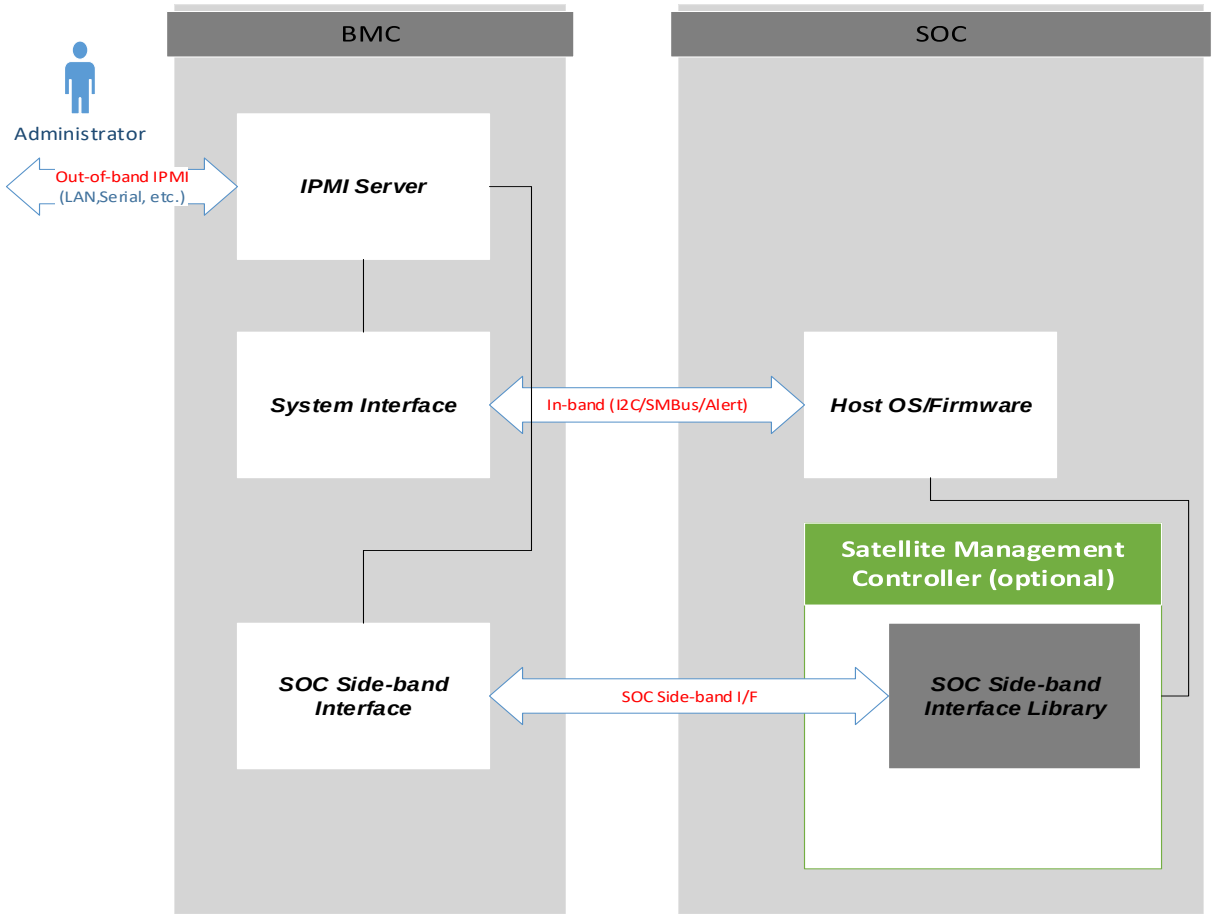
C.1 LEVEL M0

For transferring RAS error records, In-band, SOC Side-band, Out-Of-band interfaces are implementation defined.

C.2 LEVEL M1

A conceptual illustration of Required IPMI based In-band, SOC Side-band, Out-Of-band RAS interfaces for LEVEL M1 is shown in Figure C-1.

Figure C-1 IPMI based RAS Interfaces



C.2.1 SMBus System Interface (In-band Interface)

For transferring RAS error records generated in Host OS/Firmware, SBMR recommends the use of IPMI based System Interface (specifically SMBus System Interface SSIF) as the in-band interface for the Level M1 based server systems.

Other IPMI System Interfaces such as Keyboard Controller Style (KCS), System Management Interface Chip (SMIC), Block Transfer (BT) are optional and not expected to be present.

The overview of “RAS Events” interaction with the event receiver and RAS Manager through SMBus System Interface (SSIF) is illustrated in Figure C-2.

Figure C-2 IPMI based RAS Event Receiver

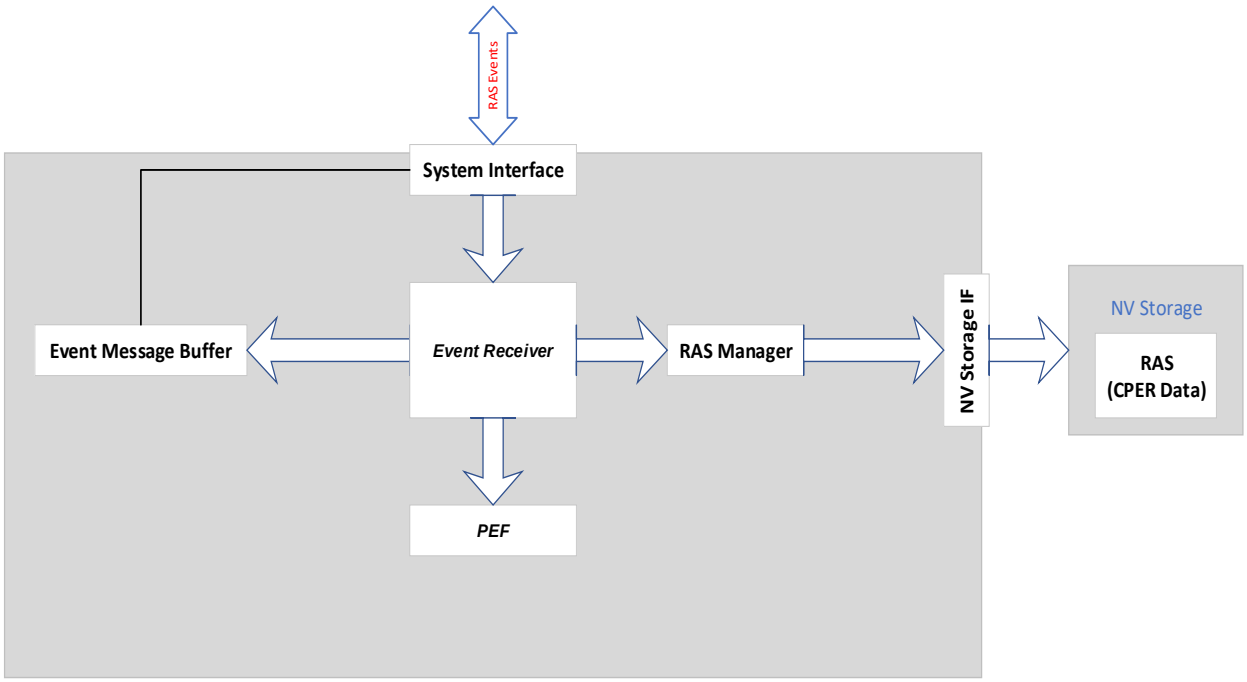


Figure C-2 represents a conceptual illustration of the way RAS event messages can be handled by a Baseboard Management Controller device that uses an external non-volatile storage device to hold the RAS Event Log. The figure shows a BMC with a shared system messaging interface where RAS Event Messages can be delivered from either firmware, SMS (system management software/OS), or an SMI/MMI Handler.

SBMR recommends creating an event type “CPER”.

When the BMC receives a message via the system interfaces, a ‘Message Handler’ function recognizes the message as being for the ‘Event’ functionality in the BMC and passes the message information on to the ‘Event Receiver’ function.

The Event Receiver function then takes the message content and issues a request to a ‘RAS Manager.’ function that formats the message as a Common Platform Error Record (CPER) Entry and calls the FLASH Interface to have the data stored.

SBMR recommends the error record data format to be in raw Common Platform Error Record CPER format when using this interface. The format of Common Platform Error Record (CPER) is defined in UEFI Specification Appendix N [4].

C.2.2 RAS IPMI Message Format

The common components of IPMI message consist of

Network Function (NetFn): A field that identifies the functional class of the message.

Request/Response identifier: A field that unambiguously differentiates Request Messages from Response Messages.

Requester’s ID: Information that identifies the source of the Request.

Responder’s ID: A field that identifies the Responder to the Request.

Command: The messages specified in this document contain a one-byte command field. Commands are unique within a given Network Function.

Data: The Data field carries the additional parameters for a request or a response, if any.

SBMR recommends the use of group extension option (as provided by IPMI specification) as it will give Arm ecosystem a broad scope for managing the transport and protocols.

Group Extensions (**2Ch, 2Dh**) - This will allow all the commands to come under a Group for Non-IPMI groups and requests.

The first data byte position in requests and responses under this network function identifies the defining body that specifies command functionality. Software assumes that the command and completion code field positions will hold command and completion code values.

"AEh" value is used to identify SBMR which is the defining body for Arm ecosystem commands.

A multi-part write is used when more than 32-bytes of IPMI message data need to be written to the BMC.

Since the size of error record format is in the order of kilo bytes, SBMG recommends the use of Multi-part write transaction.

Multi-part write transaction has one Start, multiple Middle and one End transaction.

SSIF start transaction for RAS (CPER) data will be as shown below:

Slave Address (7)	R/W (1)	SMBus CMD (8)	Length (8)	NetFn (6)	LUN (2)	IPMI CMD (8)	IPMI Data (0 or more bytes)	[PEC] (8)
	0	0x06	0x20	0x2C		0x01	0xAE Followed by CPER	

The network function code is "0x2C" and the first byte of IPMI request data is "0xAE" to indicate that the IPMI commands are defined by SBMR.

SBMR defines IPMI Command "**TransferPlatformErrorRecord**" with code "0x01" to indicate the write of multi-part CPER Record to BMC.

There can be multiple middle transactions depending on the size of Common Platform Error Record (CPER).

SSIF middle transaction for RAS (CPER) data will be as shown below:

Slave Address (7)	R/W (1)	SMBus CMD (8)	Length (8)	IPMI Data	[PEC] (8)
	0	0x07	0x20	CPER	

SSIF end transaction for RAS (CPER) data will be as shown below:

Slave Address (7)	R/W (1)	SMBus CMD (8)	Length (8)	IPMI Data	[PEC] (8)
	0	0x08		CPER	

C.2.3 SOC Side-band Interface

For transferring RAS error records either generated in Host OS/Firmware and transferred over to Satellite/Service Management Controller **or** in the Satellite/Service Management Controller itself, SOC Side-band interface for SBMR LEVEL M1 Compliant systems is implementation defined.

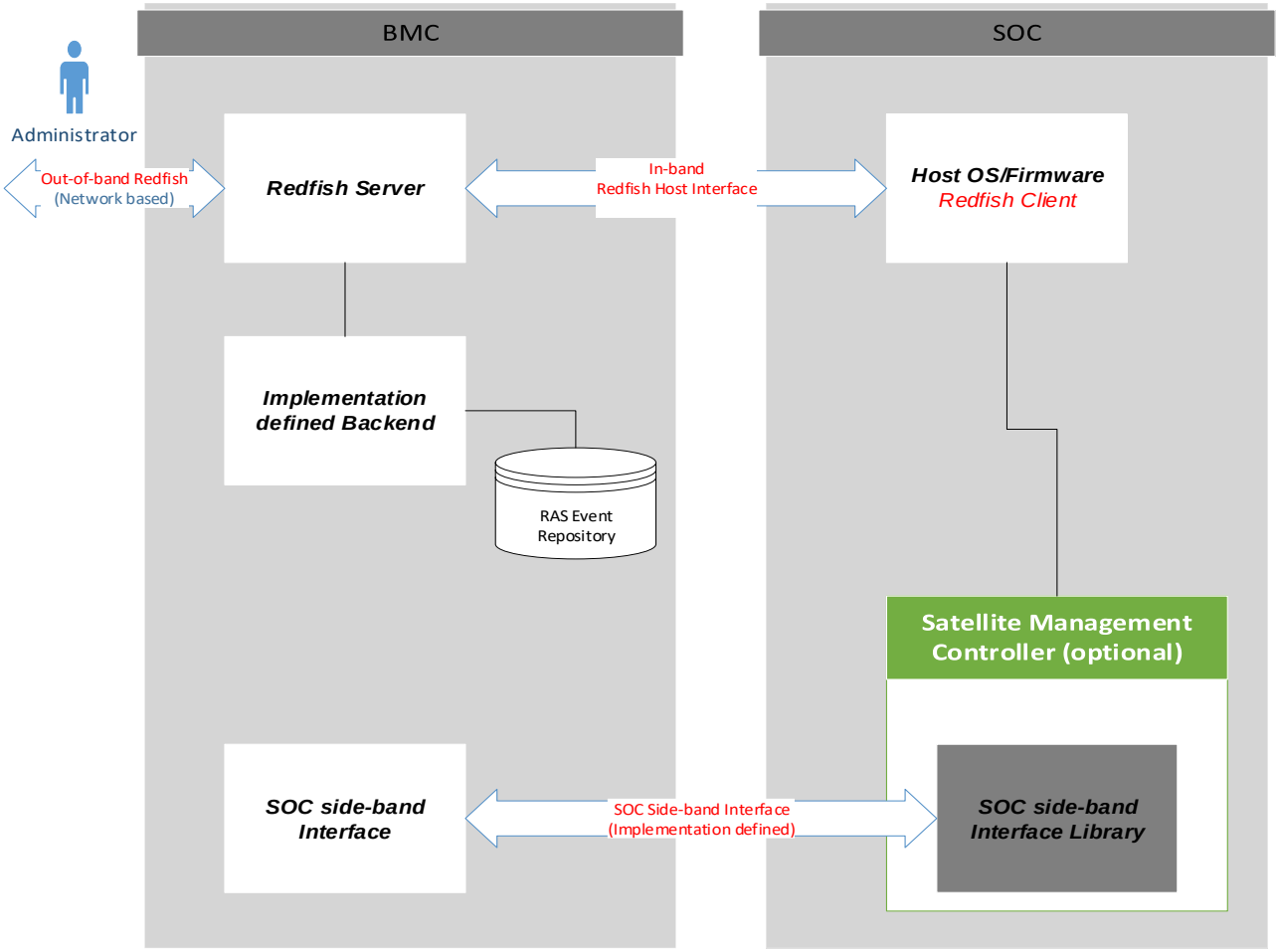
C.2.4 Out-of-band Interface

SBMR recommends IPMI based tool to extract the stored RAS error records in raw CPER format. IPMI based tool will be responsible for formatting raw CPER format data into human readable format.

C.3 LEVEL M2

LEVEL M2 requires Redfish as out-of-band interface and Redfish Host Interface as the in-band interface. A conceptual illustration of these interfaces for RAS is shown in Figure C-3.

Figure C-3 Redfish based RAS Interfaces



C.3.1 Redfish Host (in-band) Interface

For transferring RAS error records generated in Host OS/Firmware, SBMR recommends Redfish Host Interface as the in-band interface for the Level M2 based server systems.

SBMR recommends the error record data format to be in JSON format when using this interface. The error records themselves are recommended to be stored in CPER like format in the RAS Event Repository (non-volatile storage).

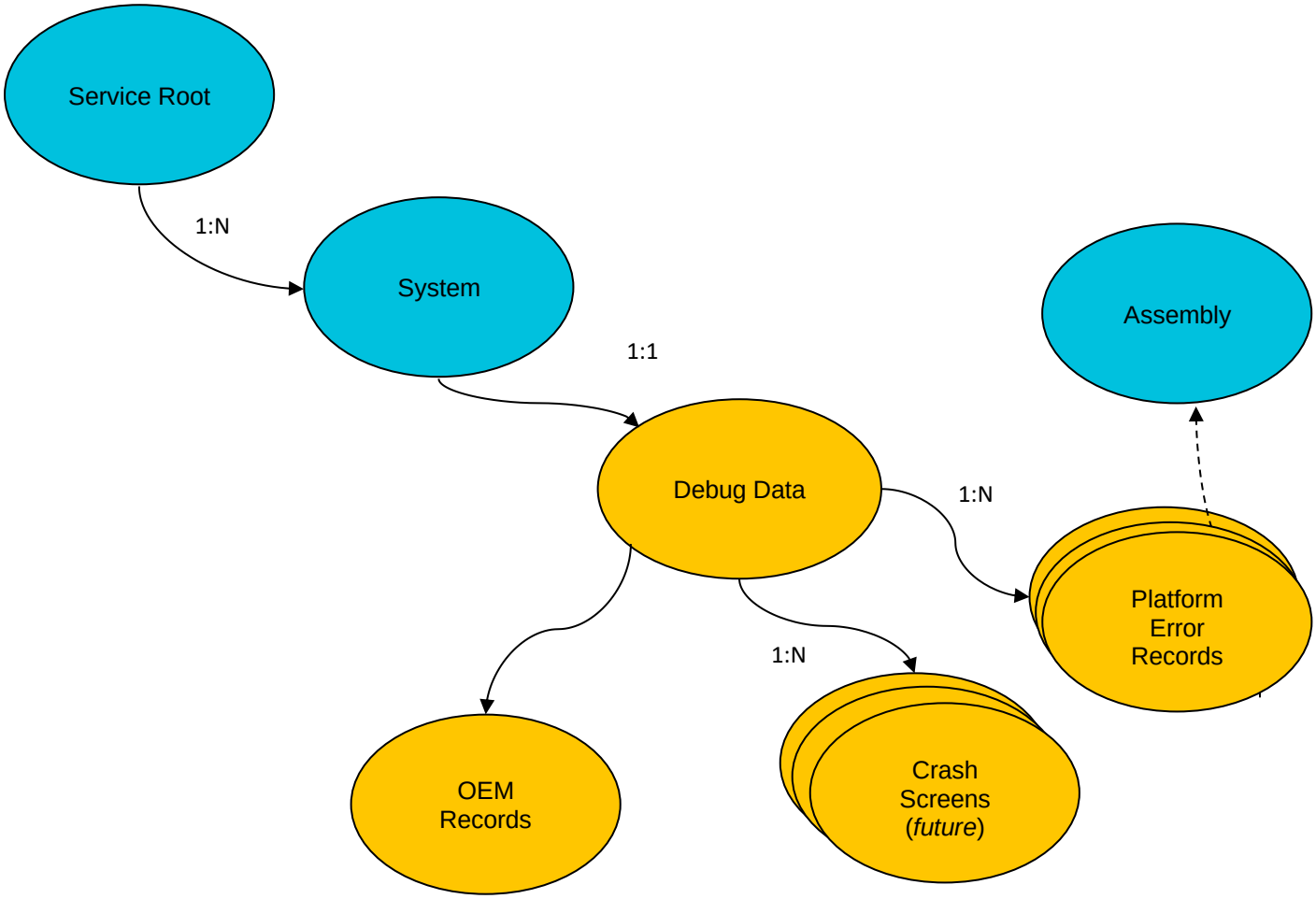
SBMR recommends that Host Interface and out-of-band API must be the same (where possible) so that client apps have minimal (if any) change to adapt.

C.3.2 RAS Redfish Message Format (proposed)

The proposed Redfish model for extracting Platform Error Records is shown below in Figure C-4.

NOTE: This is a proposal, and not a DMTF standard. This proposal is subject to change, and will be updated with the actual standard once published.

Figure C-4 Redfish Platform Error Records Proposed Model



The Redfish JSON mockup for “DebugData” is below:

```
{
    "@odata.id": "/redfish/v1/Systems/1/DebugData",
    "@odata.type": "#DebugData.v1_0_0.DebugData",
    "Id": "DebugData",
    "Name": "System Debug Data",
    "PlatformErrorRecords" :  {
          "@odata.id": "/redfish/v1/Systems/1/DebugData/PlatformErrorRecords",
     },
     "Oem": {}
}
```

The mockup for “PlatformErrorRecords” which contains an example of Memory Error is shown below. The error section itself will be in binary.

```
{
  "@odata.type": "#PlatformErrorRecords.v1_0_0.PlatformErrorRecords",
  "@odata.id": "/redfish/v1/Systems/1/PlatformErrorRecords",
  "ErrorRecords": [
    {
      "@odata.id": "/redfish/v1/Systems/1/PlatformErrorRecords#/ErrorRecords/0",
      "UefiRecordName": "HwErrRec0000",
      "BinaryDataURI": "/dumpster/debug/cper0000.bin",
      "TimeStamp": "2019-04-01T14:55:33+03:00",
      "PlatformId": "289e0c3e-5326-4a55-a378-d8f049a63699",
      "CreatorId": "ab288813-a54e-42ee-aa23-18025620c02d",
      "RecordId": "0x0123456789ABCDEF",
      "Severity": "Fatal",
      "NotificationType": "PEI",
      "RecoveredError”: false,
      "PreviousSessionError”: false,
      "SimulatedError”: true,
      "Sections”: [
        {
          "Name": "Memory Errors",
          "Description”: "Memory Slot 1 Error",
          “SectionType": “Memory",
          "Severity": "Fatal",
          "PrimarySection”: true,
          "Assembly”: {
            "@odata.id": "/redfish/v1/Systems/1/Memory/1/Assembly#/Assemblies/0"
           },
        }
      ],
    }
  ],
  "Oem": {},
}
```

C.3.3 SOC-sideband Interface

For transferring RAS error records either generated in Host OS/Firmware and transferred over to Satellite/Service Management Controller **or** in the Satellite/Service Management Controller itself, SOC Side-band interface for SBMR LEVEL M2 Compliant systems is implementation defined.

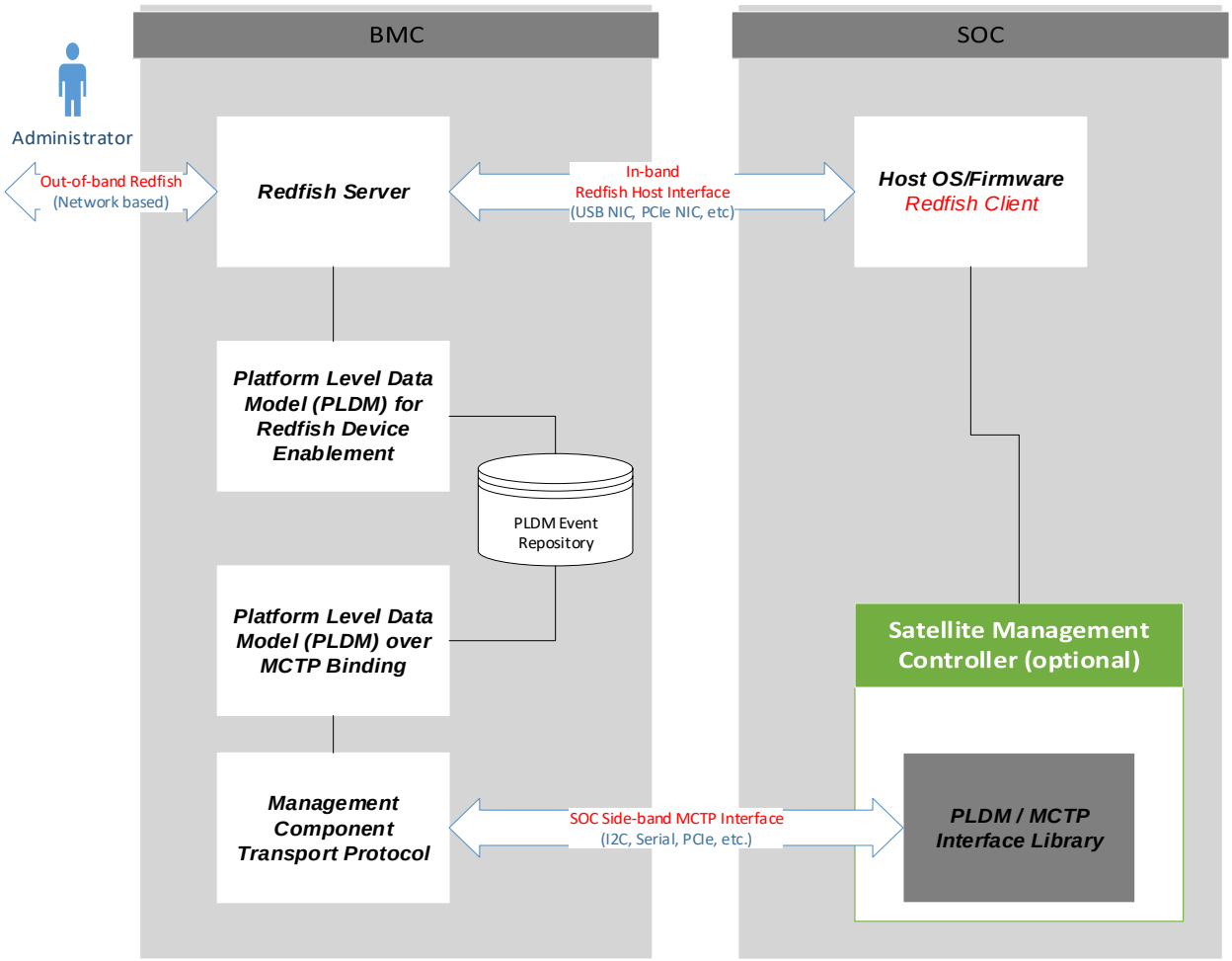
C.3.4 Out-of-Band Interface

SBMR recommends Redfish based tool to extract the stored RAS error records in CPER like format from RAS event repository. Backend translates error record from CPER like format to JSON format.

C.4 LEVEL M3a/M4a

LEVEL M3 adds the additional requirement of MCTP based SOC side-band interface in addition to Redfish as out-of-band interface and Redfish Host Interface as the in-band interface. A conceptual illustration of these interfaces for RAS is shown in Figure C-5.

Figure C-5 Redfish/PLDM/MCTP based RAS Interfaces



C.4.1 Redfish Host (in-band) Interface

For transferring RAS error records generated in Host OS/Firmware, the recommendations are same as Level M2 based server systems.

C.4.2 MCTP (SOC side-band) Interface

For transferring RAS error records either generated in Host OS/Firmware and transferred over to Satellite/Service Management Controller **or** in the Satellite/Service Management Controller itself, SBMR recommends

Management Component Transport Protocol (MCTP) as the SoC side-band transport layer interface for the Level M3a/M4a based server systems.

The physical binding of MCTP is left best to System Implementors. Some examples of the physical binding include MCTP over I_2C, MCTP over PCIe VDM, MCTP over Serial.

Further, SBMR recommends Platform Level Data Model (PLDM) as the SOC side-band message definition and data layer interface for the Level M3a/M4a based server systems.

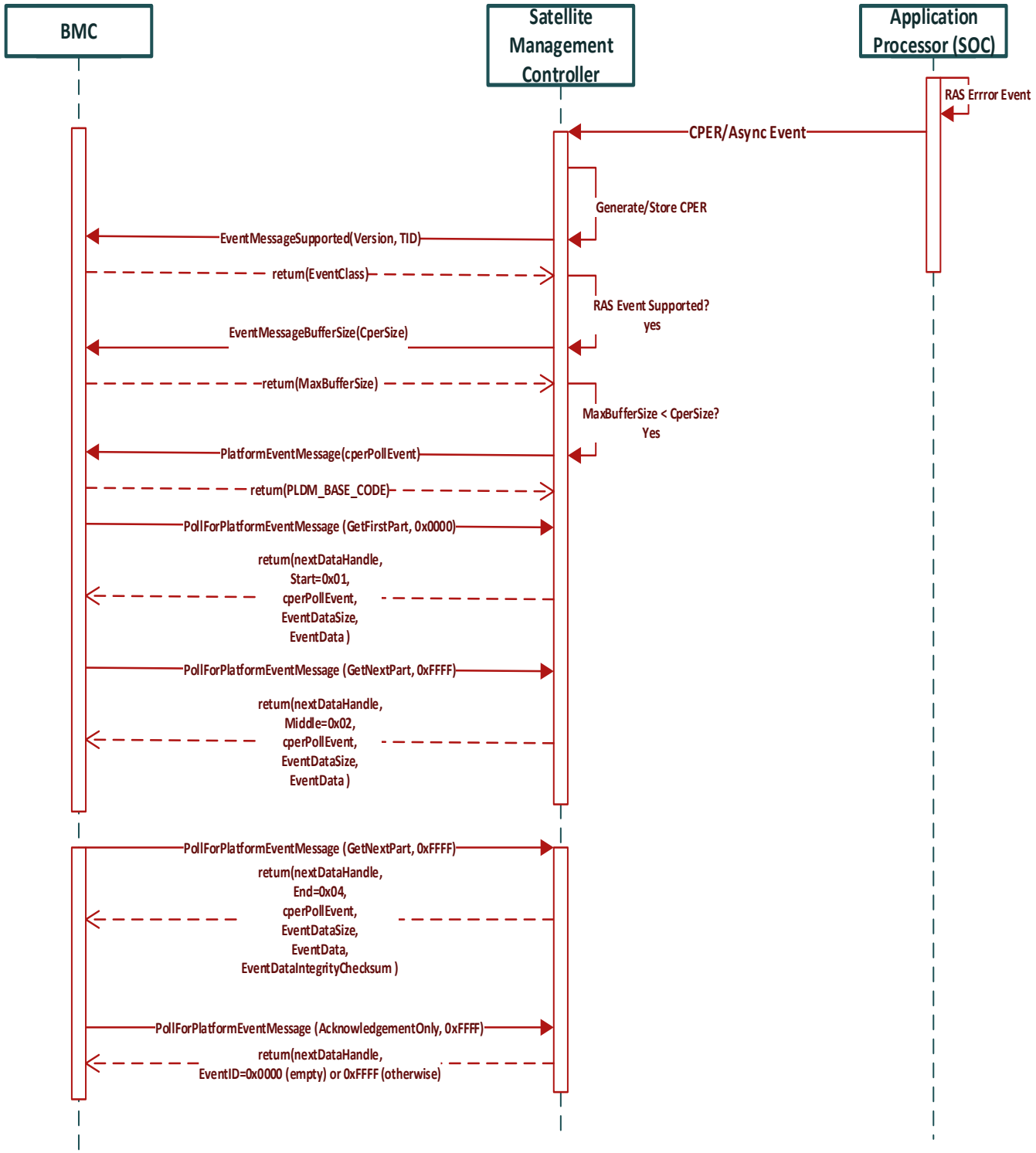
SBMR recommends the error record data format to be in CPER like format when using this interface.

SBMR recommends the use of "***PlatformEventMessage***", "***PollForPlatformEventMessage***", "***EventMessageSupported***" and "***EventMessageBufferSize"*** APIs/Commands to transfer CPER like format RAS errors from the Satellite/Service Management Controller to the BMC. A new event class "***cperPollEvent***" event class is proposed to enable this feature.

For more details, please refer to PLDM for Platform Monitoring and Control Specification [20]. Figure C-6 shows an example flow that demonstrates switching to polled event transfer to receive an CPER event with large event data.

When BMC gets a "***cperPollEvent***", this is a signal that an event with a large amount of cper data is next to be transferred. The BMC then uses the ***PollForPlatformEventMessage*** command with ***TransferOperationFlag*** set to GetFirstPart to initiate the transfer. In response, the satellite management controller supplies the first chunk of data along with a transfer handle for the next portion and a *transferFlag* of Start, which indicates that this is the first chunk and there is at least one more. The BMC then retrieves the next chunk in the same fashion, using the *nextDataTransferHandle* supplied in the previous response. So long as the response message transferFlag field is set to Middle, the BMC knows that more data is waiting to be retrieved, and repeats this process using the most recently received *nextDataTransferHandle* to obtain the next data chunk each time. Finally, when the *transferFlag* comes back as End, the BMC knows the transfer is complete and can verify the *eventDataIntegrityChecksum* against the reassembled cper event data. Assuming the transfer was successful, the BMC can now acknowledge receipt of the event and switch back to asynchronous transfer of events by sending a final ***PollForPlatformEventMessage*** command with *TransferOperationFlag* set to AcknowledgementOnly.

Figure C-6 Redfish/PLDM/MCTP based RAS Interfaces



C.4.3 RAS PLDM Message Format

The proposed RAS/CPER PLDM event log entry format is shown in Table C-1.

Table C-1 RAS/CPER PLDM event log entry format

Byte	Type	Field
0	enum8	**entryType** value: {**PLDMPlatformEvent**, OEMTimestampedEntry, OEMEntry }
1	uint8	**entryDataLength** The size in bytes of the entryData field.
variable	–	**entryData** Data for the entry, dependent on the entryType. entryType = PLDMPlatformEvent for CPER/RAS

The proposed entryData format for RAS/CPER PLDM event is shown in Table C-2.

Table C-2 RAS/CPER PLDM event log entryData format

Byte	Type	Field
0	sint8	**entryTimestampUTCOffset** The UTC offset for the log entry timestamp in increments of 1/2 hour special value: 0xFF = unspecified
1:5	uint40	**entryTimestampSeconds** This value corresponds to a 40-bit unsigned integer that represents the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds).
6	uint8	**entryTimestamp100s** This value provides a number of 1/100ths of a second added to entryTimestampSeconds. value: 0 to 99 special value: 0xFF = unspecified. Use this value if the implementation timestamps entries to no finer than a one-second resolution.
variable	–	**eventData** The eventData format is same as the response of the PollforPlatformEventMessage command.

The proposed eventData format for RAS/CPER PLDM event is shown in Table C-3.

Table C-3 RAS/CPER PLDM eventData format

Byte	Type	Field
0:1	uint16	**sectionCount** This field indicates the number of valid sections associated with the record, corresponding to section descriptors.

2:5	uint32	**errorSeverity** 0 - Recoverable (also called non-fatal uncorrected) 1 - Fatal 2 - Corrected 3 - Informational All other values are reserved.
6:9	uint32	**flags** Flags field contains information that describes the error record. HW_ERROR_FLAGS_RECOVERED = 1 HW_ERROR_FLAGS_PREVERR = 2 HW_ERROR_FLAGS_SIMULATED = 4
variable	-	**sectionDescriptor** This field describes error section description.

The proposed sectionDescriptor format for RAS/CPER PLDM event is shown in Table C-4.

Table C-4 RAS/CPER PLDM sectionDescriptor format

Byte	Type	Field
0:15	uint128	**sectionType** This field holds a pre-assigned GUID value indicating that it is a section of a particular error.
16:31	uint128	**fruId** GUID representing the FRU ID. The default value is zero indicating an invalid FRU ID. This can be used to uniquely identify a physical device for tracking purposes. Association of a GUID to a physical device is implementation defined.
32:35	uint32	**sectionSeverity** This field indicates the severity associated with the error section. 0 – Recoverable (also called non-fatal uncorrected) 1 – Fatal 2 – Corrected 3 – Informational All other values are reserved.
36:55	uint160	**fruString** ASCII string identifying the FRU hardware.
variable		**section** section consists of error information.

The format of a section is identified by the GUID populated in the Section Descriptor's "*sectionType*" field and is outside the scope of this document.

For more details on standard and non-standard sections, please refer to UEFI Specification Appendix N [4].

C.4.4 Out of Band Interface

SBMR recommends Redfish based tool to extract the stored RAS error records in CPER like format from PLDM event repository. PLDM backend translates error record from CPER like format to JSON format. For more details, please refer to PLDM for Redfish Device Enablement specification [24].

APPENDIX D PLATFORM MONITORING AND CONTROL IMPLEMENTATION GUIDE

D.1 Introduction

Managed entity refers to the physical or logical entity that is being managed through management parameters. Examples of physical entities include fans, processors, power supplies, circuit cards, chassis, and so on. Examples of logical entities include virtual processors, cooling domains, system security states, and so on.

D.2 IPMI Commands to Monitor and Control Managed entities

SBMR recommends the following list of IPMI commands which aid in monitoring and control of managed entities.

1. *Get Sensor Reading*
2. *Get Sensor Reading Factors*
3. *Set Sensor Hysteresis*
4. *Get Sensor Hysteresis*
5. *Set Sensor Thresholds*
6. *Get Sensor Thresholds*
7. *Set Sensor Event Enable*
8. *Get Sensor Event Enable*
9. *Re-arm Sensor Events*
10. *Get Sensor Event Status*
11. *Set Sensor Type*
12. *Get Sensor Type*
13. *Set Sensor Reading and Event Status*

For more details, please refer to Intelligent Platform Management Interface v2.0 (IPMI) specification [26].

Sensor Data Records (SDRs)

SBMR recommends SDR Type 01h, Full Sensor Record to describe the managed entities. For more details, please refer to Intelligent Platform Management Interface v2.0 (IPMI) specification [26].

SBMR recommends the following list of IPMI commands which aid in management of Sensor Data Records (SDRs) of managed entities.

1. *Get Device SDR Info*
2. *Get Device SDR*
3. *Reserve Device SDR Repository*
4. *Get SDR Repository Info*
5. *Get SDR*
6. *Add SDR*
7. *Partial Add SDR*
8. *Clear SDR Repository*

Sensor Data Records (SDRs) are data records that contain information about the type and number of managed entities in the platform, sensor threshold support, event generation capabilities, and information on what types of readings the sensor provides.

The general Sensor Data Record format consists of three major components, the Record Header, Record ‘Key’ fields, and the Record Body.

Sensor Type Code, Offset and Unit

SBMR recommends the use of Sensor Type values and sensor-specific event offsets (if any) as defined by Intelligent Platform Management Interface (IPMI) specification for managed entities. For more details on the Sensor Type values, please refer to Intelligent Platform Management Interface (IPMI) specification Table 42-3 [26].

For a list of sensor unit codes, please refer to Intelligent Platform Management Interface (IPMI) Specification Table 43-15 [26].

Entity IDs

SBMR recommends the use of Entity IDs which identifies the sensor association with a physical container. SBMR carves out Entity IDs to identify SOC firmware (E.g., pre-EFI firmware), SOC Management Software (E.g., Satellite/Service Management Software) from OEM System Integrator defined range 0xD0 – 0xFF as defined in the table below.

Code	Entity	Comments
0xE0	SOC Management Software	This value identifies firmware or software running on a satellite/service management controller within/outside Arm SOC.
0xE1	SOC firmware	This value identifies pre-EFI firmware on Arm SOCs.

For a complete list of entity IDs, please refer to Intelligent Platform Management Interface (IPMI) Specification Table 43-13 [26].

D.3 Redfish Schema to Monitor and Control Managed entities

SBMR recommends the use of the schema for sensor as defined by DMTF here [10][7]:
https://redfish.dmtf.org/schemas/v1/Sensor.v1_0_1.json

D.4 PLDM Commands/APIs to Monitor and Control Managed entities

SBMR recommends the following list of PLDM commands which aid in monitoring and control of SOC connected Numeric and State managed entities/effecters:

1. *SetNumericSensorEnable*
2. *GetSensorReading*
3. *InitNumericSensor*
4. *SetStateSensorEnables*
5. *GetStateSensorReadings*
6. *InitStateSensor*
7. *SetNumericEffecterEnable*
8. *SetNumericEffecterValue*
9. *GetNumericEffecterValue*
10. *SetStateEffecterEnables*
11. *SetStateEffecterStates*
12. *GetStateEffecterStates*

Platform Descriptor Records (PDRs)

SBMR recommends the use of Platform Descriptor Records (PDRs).

SBMR recommends the following list of PLDM commands which aid in management of Platform Descriptor Records (PDRs) of managed entities:

1. *GetPDRRepositoryInfo*
2. *GetPDR*
3. *RunInitAgent*

For more details on the PLDM Commands, please refer to PLDM for Platform Monitoring and Control Specification [20].

PDRs provide semantic information for managed entities.

APPENDIX E REFERENCE IMPLEMENTATION OF BMC REMOTE DEBUG SOLUTION USING OPENOCD

E.1 Introduction

BMC Remote debug is the act of gaining visibility and control of the hardware and software behaviors of a Server SoC, using a debug client which is not directly connected to the Server SoC, but connected to a debug server running on a baseboard manageability controller (BMC).

E.2 LEVEL M1/M2

This section describes a reference solution for implementing BMC remote debug using OpenOCD (http://openocd.org/) for SBMR LEVEL M1/M2 Compliant Servers.

This reference solution for BMC remote debug integrates open source OpenOCD inside the open source OpenBMC stack. OpenOCD implements support for Arm Debug Interface debugging architecture.

OpenOCD includes in-built JTAG controller drivers which need to be compiled in to the OpenOCD binary to support a specific JTAG controller. Support for a new JTAG controller can be added by writing a new driver.

OpenOCD provides one of these TCP/IP port-based interface for communication:

1. Gdb port (default port : 3333)
2. Tcl port (default port : 6666)
3. Telnet port (default port : 4444)

A reference implementation of remote debug feature using GNU MCU Eclipse plugin, OpenOCD using JTAG interface is shown in Figure E-1.

Figure E-1 Reference implementation of remote debug.

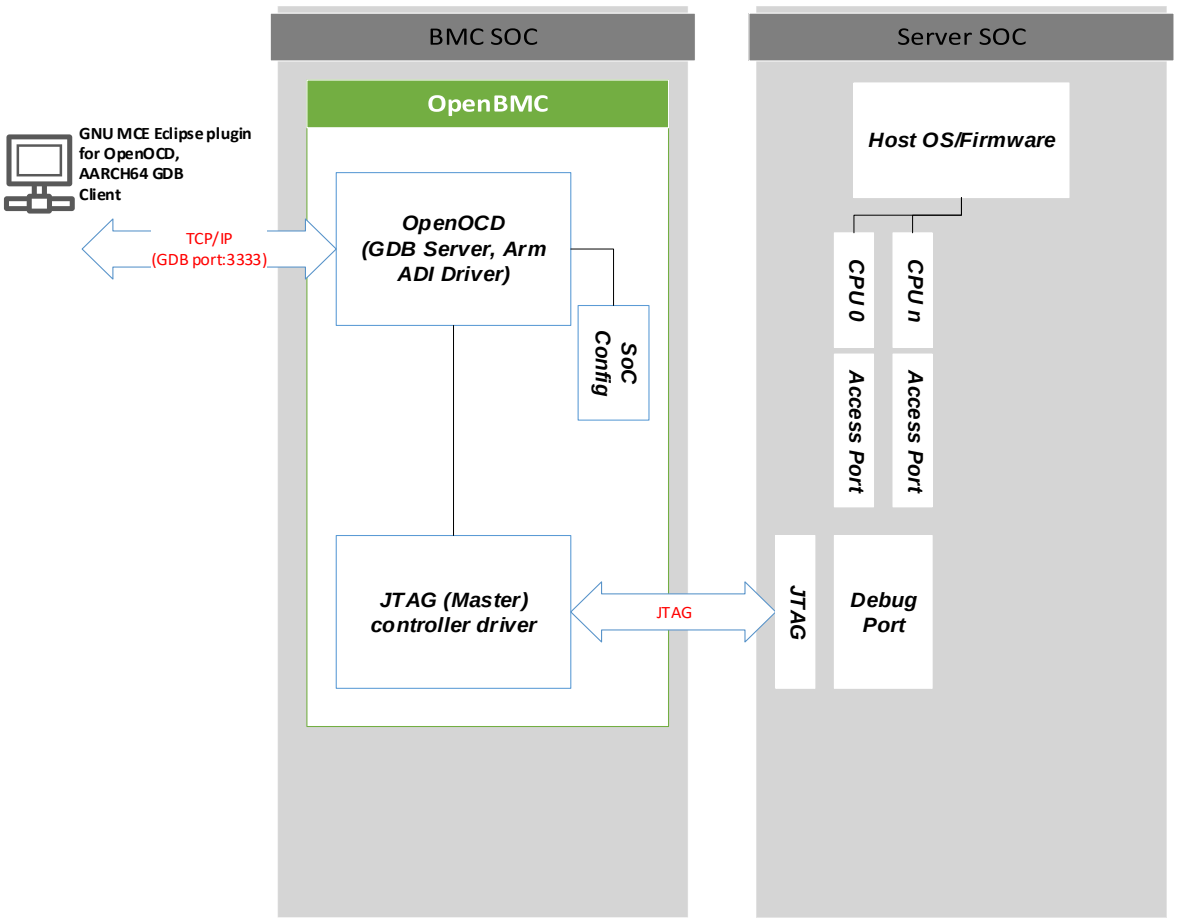


Figure E-1 illustrates the source level bare metal debug of Server SOC firmware and kernel debug from a GDB client running on the remote machine connected to OpenOCD GDB Server running on the BMC. OpenOCD includes a JTAG controller (master) driver for the BMC platform, which aids in communication with the Server SOC Arm Debug Interface.

User/Administrator can use Graphical User Interface (GUI) based integrated development environment (IDE) Eclipse which supports OpenOCD via the GDB Hardware Debugging plug-in. OpenOCD GDB remote debug Server running on baseboard manageability controller (BMC) listens on port 3333 for OpenOCD aware GDB debug client connections. OpenOCD also requires the SOC configuration of the system under debug which should provide hardware specific details. For more information, refer to OpenOCD user guide [40].

User/Administrator can now access the debug functions remotely through the BMC including but not limited to:

- Full memory and register access
- run and stop
- software and hardware breakpoints and watchpoints
- target reset (restart)
- binary program downloading
- step-over-range
- single stepping