

Arm[®] Server Base Manageability Requirements 1.1

Platform Design Document

Non-confidential



Release information

Date	Version	Changes
11 Feb 2021	Issue C	<ul style="list-style-type: none">• SBMR 1.1 release• Add compliance level M2.1• Add standard Boot Progress Code feature• Clarify IPMI SSIF support• Miscellaneous typos, clarifications, and editorial changes
15 Jun 2020	Issue B	<ul style="list-style-type: none">• License LES-PRE-21585
30 Jan 2020	Issue A	<ul style="list-style-type: none">• Initial release, SBMR 1.0

Arm Non-Confidential Document Licence (“Licence”)

This Licence is a legal agreement between you and Arm Limited (“**Arm**”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this Licence (“**Document**”). Arm licenses its intellectual property in 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.

“**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.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENCE, TO THE FULLEST EXTENT PERMITTED 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.

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 Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. No licence, express, implied or otherwise, is granted to Licensee under this Licence, to use the Arm trade marks in connection with the Document or any products based thereon. Visit Arm's website at <http://www.arm.com/company/policies/trademarks> for more information about Arm's trademarks.

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

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-21585 version 4.0

Copyright © 2021 Arm Limited. All rights reserved.

Contents

Release information	2
Arm Non-Confidential Document Licence (“Licence”)	3
About this document	8
Terms and abbreviations	8
References	9
Cross References	10
Feedback	10
1 SCOPE AND BACKGROUND	11
1.1 Scope	11
1.2 Background	12
1.2.1 Host/SoC in-band interface	12
1.2.2 SoC side-band interface	13
1.2.3 PCIe connection between the Arm SoC and the BMC	13
1.2.4 USB connection between the Arm SoC and the BMC	13
1.2.5 JTAG connection between the Arm SoC and the BMC	13
1.2.6 Additional connectivity between the Arm SoC and the BMC	14
1.2.7 Multi-socket platform	14
1.3 Arm SoC-BMC interface terminology	14
2 COMPLIANCE LEVELS AND REQUIREMENTS	16
2.1 Level M0	17
2.2 Level M1	18
2.2.1 SoC-BMC interface	19
2.2.2 BMC-platform elements interface recommendations	20
2.2.3 BMC management services (out-of-band) interface recommendation	20
2.3 Level M2	21
2.3.1 SoC-BMC interfaces	22
2.3.2 BMC-platform elements interface	22
2.3.3 BMC-IO device interface	22
2.3.4 BMC management services (out-of-band) interface	23
2.4 Level M2.1	24
2.4.1 SoC-BMC interfaces	25
2.4.2 BMC-platform elements interface	25
2.4.3 BMC-IO device interface	26
2.4.4 BMC management services (out-of-band) interface	26
2.5 Level M3 alpha (work-in-progress)	27
2.5.1 Requirements	28
2.5.2 SoC-BMC interface	28
2.5.3 BMC-platform elements interface recommendations	28
2.5.4 BMC-IO device interface recommendations	28
2.6 Level M4 alpha (work-in-progress)	29
2.6.1 Requirements	30
2.6.2 SoC-BMC interface	30
2.6.3 BMC-platform elements interface recommendations	30
2.6.4 BMC-IO device interface recommendations	30
A OPENBMC	32
B IPMI IMPLEMENTATION GUIDE	33
B.1 Standard IPMI commands	33
B.1.1 Remote power control	33
B.1.2 Boot device selection	33

B.1.3	BMC to Host mapping	33
B.1.4	BMC user manipulation	34
B.1.5	Redfish host interface credentials bootstrapping	34
B.1.6	IPMI support verification	34
B.2	Arm standard IPMI commands	34
B.2.1	General IPMI commands format	34
B.2.2	List of Arm standard IPMI commands	35
B.3	IPMI specification clarifications and corrections	35
B.4	SSIF single and multi-part transactions	36
C	RAS MESSAGE FORMAT	38
C.1	Level M0	38
C.2	Level M1	38
C.2.1	SMBus System Interface (SSIF) in-band interface	39
C.2.2	RAS IPMI message format	40
C.2.3	SoC side-band interface	40
C.2.4	Out-of-band interface	40
C.3	Level M2 and Level M2.1	40
C.3.1	Redfish and IPMI host (in-band) interfaces	41
C.3.2	RAS Redfish message format	41
C.3.3	SoC side-band interface	42
C.3.4	Out-of-band interface	42
C.4	Level M3a and M4a	42
C.4.1	Redfish host (in-band) interface	43
C.4.2	MCTP (SoC side-band) interface	43
C.4.3	RAS PLDM message format	45
C.4.4	Out-of-band interface	47
D	PLATFORM MONITORING AND CONTROL IMPLEMENTATION GUIDE	48
D.1	Introduction	48
D.2	IPMI commands to monitor and control managed entities	48
D.3	Redfish schema to monitor and control managed entities	49
D.4	PLDM commands/APIs to monitor and control managed entities	49
E	REFERENCE IMPLEMENTATION OF BMC REMOTE DEBUG SOLUTION USING OPENOCD	51
E.1	Introduction	51
E.2	Levels M1, M2, M2.1	51
F	BOOT PROGRESS CODE REPORTING	53
F.1	IPMI commands for boot progress codes	53
F.1.1	Send boot progress code (NetFn 2Ch, Command 02h)	53
F.1.2	Get boot progress code (NetFn 2Ch, Command 03h)	53
F.2	Boot progress code format	54
F.2.1	Example progress codes (IPMI)	55
F.2.2	Example boot progress codes (Redfish)	57
F.3	Common boot progress codes	58

About this document

This document is intended for SBSA [1] -compliant 64-bit Arm based servers. It provides a path to establish a common foundation for server management, where common capabilities are standardized, and differentiation truly valuable to the end-users are built on top.

This specification leverages the prevalent industry standard system management specifications of Redfish[2], Platform Level Data Model (PLDM)[3] and Management Component Transport Protocol (MCTP)[4]. These specifications are defined in the DMTF Redfish Forum and Platform Management Components Intercommunication (PMCI) Working Group.

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
ACPI	Advanced Configuration and Power Interface.
BMC	Baseboard Management Controller. 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.
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 pre-boot 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.
Node	For the purpose of this specification, a node is a single server system in a group of managed servers.
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.

Term	Meaning
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.
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.
SiP	Silicon Partner. In this document, the silicon manufacturer.
UEFI	Unified Extensible Firmware Interface.
UEFI Boot Services	Functionality that is provided to an Operating System boot loader or UEFI application before the ExitBootServices() call.
UEFI Runtime Services	Functionality that is provided to an Operating System after the ExitBootServices() call.

References

This section lists publications by Arm and by third parties.

See Arm Developer (<http://developer.arm.com>) for access to Arm documentation.

- [1] *DEN 0029 Server Base System Architecture SBSA*. Arm Ltd.
- [2] *DSP0266 Redfish Specification*. DMTF.
- [3] *DSP0240 PLDM Base Specification*. DMTF.
- [4] *DSP0236 MCTP Base Specification*. DMTF.
- [5] *Advanced Configuration and Power Interface Specification*. UEFI.org.
- [6] *Unified Extensible Firmware Interface Specification*. UEFI.org.
- [7] *DSP8010 Redfish Schema*. DMTF.
- [8] *Intelligent Platform Management Interface 2.0, Revision 1.1 (October 2013)*. Dell, HP, Intel, NEC.
- [9] *OCP Baseline Hardware Management Redfish Profile*. Open Compute Project.
- [10] *OCP Server Hardware Management Redfish Profile*. Open Compute Project.
- [11] *DSP0134 System Management BIOS (SMBIOS) Reference Specification*. DMTF.
- [12] *DSP0256 MCTP Host Interface Specification*. DMTF.
- [13] *DSP0270 Redfish Host Interface Specification*. DMTF.
- [14] *DSP0237 MCTP PCIe VDM Transport Binding Specification*. DMTF.
- [15] <https://www.opencompute.org/wiki/Server/SpecsAndDesigns>. Open Compute Project.
- [16] *DEN 0044 Server Base Boot Requirements, System Software on Arm® Platforms*. Arm Ltd.
- [17] *Server Base Security Guide (SBSG)*. Arm Ltd.
- [18] *Arm IHI 0031 Arm Debug Interface Architecture Specification, ADIv5*. Arm Ltd.
- [19] *Arm IHI 0074 Arm Debug Interface Architecture Specification, ADIv6*. Arm Ltd.

- [20] *DSP0222 NC-SI Specification*. DMTF.
- [21] *DSP0272 Redfish Interoperability Profile Specification*. DMTF.
- [22] *DSP8013 Redfish Interoperability Profiles Bundles*. DMTF.
- [23] *DSP0245 PLDM IDs and Codes Specification*. DMTF.
- [24] *DSP0248 PLDM for Platform Monitoring and Control Specification*. DMTF.
- [25] *DSP0249 PLDM State Set Specification*. DMTF.
- [26] *DSP0267 PLDM for Firmware Update Specification*. DMTF.
- [27] *DSP0239 MCTP IDs and Codes*. DMTF.
- [28] *DSP0241 PLDM Over MCTP Binding Specification*. DMTF.
- [29] *DSP0237 MCTP SMBus/I2C Transport Binding Specification*. DMTF.
- [30] *DSP0261 NC-SI over MCTP Binding Specification*. DMTF.
- [31] *DSP0218 PLDM for Redfish Device Enablement Specification*. DMTF.
- [32] *DSP0235 NVMe over MCTP Binding Specification*. DMTF.
- [33] *DSP0268 Redfish Schema Supplement*. DMTF.
- [34] <http://openocd.org/doc-release/pdf/openocd.pdf>. OpenOCD Project.
- [35] *Platform Initialization Specification*. UEFI.org.
- [36] *DDI 0487 Arm® Architecture Reference Manual ARMv8, for the ARMv8-A architecture profile*. Arm Ltd.

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 [5] section 5.6.6 - UEFI § 6.1 - Reference to the UEFI specification [6] section 6.1

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 Requirements).
- The document ID and version (DEN0069C 1.1).
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 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 differentiation truly valuable to the end-users are built on top.

1.1 Scope

Redfish [2], PLDM [3], and MCTP [4] 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 equally usable by apps, UIs and scripts. Redfish resources are schema-backed and human readable, with schemas [7] defined using JSON Schema, OData 4.0, or OpenAPI formats. Redfish provides a secure, multi-node capable replacement for IPMI-over-LAN [8]. It is intended to meet Open Compute Project (OCP) [9][10] remote machine management requirements.

The support for the legacy Intelligent Platform Management Interface (IPMI)[8] is still required as IPMI-based tools are still widely used by the end-users. 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.

This document addresses the need to establish the following common standard interface sets (See Figure 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 1.2.
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

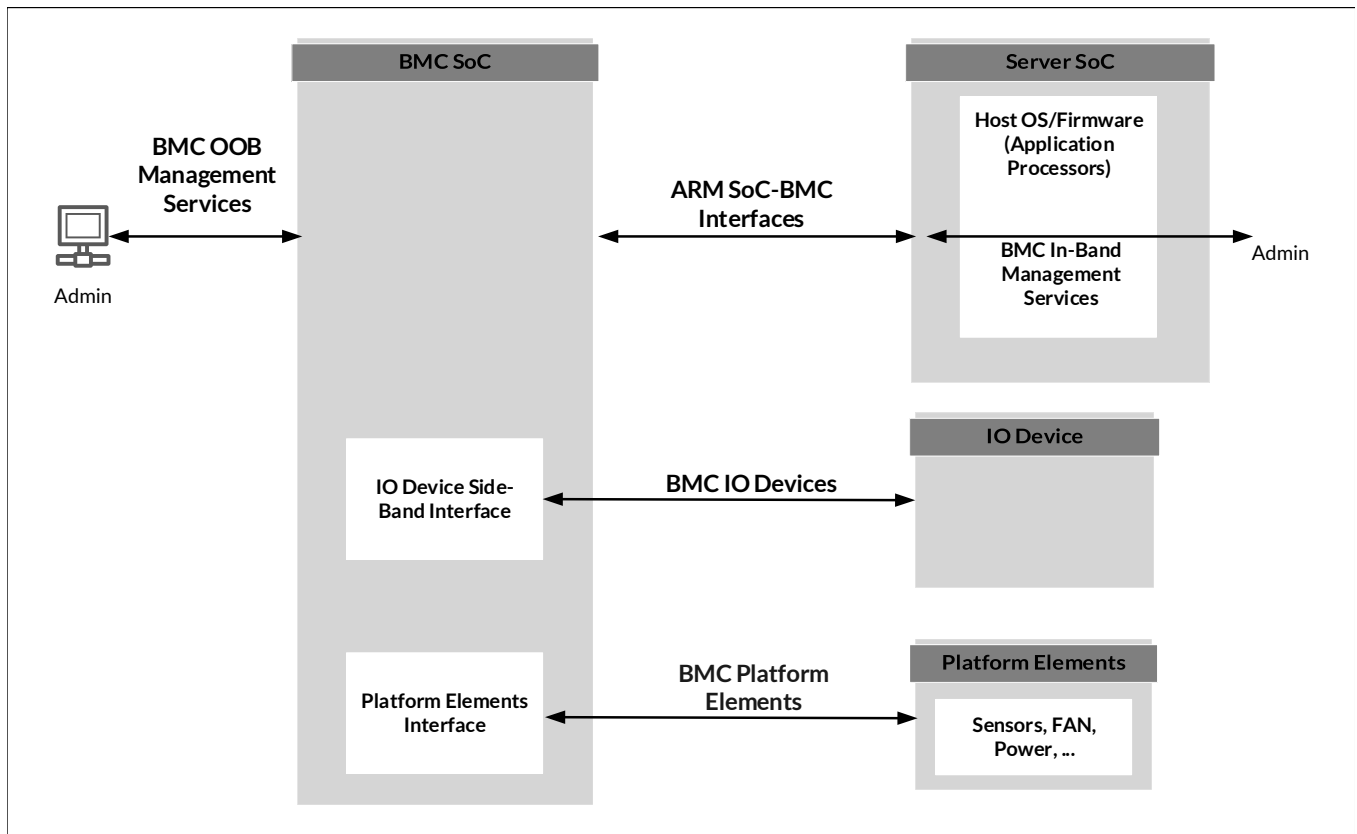


Figure 1: Server Management Interfaces

The focus of this document is to provide manageability requirements for various SBMR Mx compliance levels, as described in Section 2. These are requirements with respect to relevant interfaces between the Arm SoC and the BMC, as described in the Table 3 summary below.

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

1.2 Background

There are several interfaces used for communication and interaction between the Arm SoC and the BMC.

1.2.1 Host/SoC in-band interface

This interface is used by the Host Software, such as OS, Hypervisor, User Software, as well as System Firmware, such as UEFI [6], to communicate with the BMC. It is typically exposed to Host Software via SMBIOS [11], ACPI [5] tables (e.g. SPMI), and/or PCIe configuration space. Earlier Arm server systems are IPMI based, with newer Arm server systems transitioning to Redfish [2] and MCTP host interface [12].

Typical use cases of this interface include:

- UEFI - BMC communication, using IPMI on earlier Arm server systems, and MCTP host interface on later Arm server systems:
 - Reporting SMBIOS [11] table
 - Reporting boot progress codes

- Error reporting
- General event logging
- OS/Hypervisor software – BMC communication
 - Redfish Authentication, using IPMI on earlier Arm server systems, possibly MCTP or other interfaces on later Arm server systems.
- User software – BMC communication
 - User or Admin access to BMC management services, using IPMI and/or Redfish Host Interface [13], for local server configuration, update, deployment, or monitoring.

1.2.2 SoC side-band interface

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

- Early stages of boot progress codes reporting
- Telemetry, such as Temperature, power,)
- RAS error reporting
- Early stages of boot event logging

1.2.3 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 using PCIe Vendor Defined Messages (VDM) path [14]. 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.

1.2.4 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 or integrated in the Arm SoC. It could be an external onboard PCIe-based USB controller or PHY that connects to the BMC USB ports.

1.2.5 JTAG connection between the Arm SoC and the BMC

This interface may exist for the following use-cases :

- Remote hardware debug, such as breakpoints and single stepping, 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.

1.2.6 Additional connectivity between the Arm SoC and the BMC

Various physical media interfaces may exist between the Arm SoC and the BMC 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

1.2.7 Multi-socket platform

A multi-socket system is a Server system containing two or more SoCs operating coherently and running a single OS/hypervisor. In such a system, OS owned interfaces, such as the IPMI host interface, the Redfish host interface, and video console re-direction, must exist as one per system, unless otherwise stated in this specification.

1.3 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 3: 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), Redfish Host Interface (USB/PCIe network), or MCTP Host Interface. 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. The 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 over the management network	Yes
SoC Debug Interface (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, such as critical thermal events or SoC errors. NOTE: Some pins may be bi-directional (e.g. PROCHOT)	Partially Covered

Name	Master	Slave	Description / Example / Notes	In SBMR Scope?
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, such as critical thermal events or SoC errors. NOTE: Some pins may be bi-directional (e.g. PROCHOT)	Partially Covered
Serial Console (UART)	SoC	BMC	Used for implementing Serial-over-LAN (SoL). Arm SoC typically have at least one or more UARTs. Must be an Arm SBSA [1] compliant UART controller on the SoC side. Default Baud rate for interoperability with commercially available BMCs is required to be 115200 bits/second.	Yes
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, such as 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, such as 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 for power supplies, voltage regulators, platform sensors, and other platform components NOTE: These interfaces are outside the scope of the SBMR compliance requirements. Some recommendations and guidance may be provided based on external specifications and standards.	No

2 COMPLIANCE LEVELS AND REQUIREMENTS

This specification defines a number of levels of manageability compliance 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 [1] levels.

This specification defines a set of requirements and recommendations for each compliance level. The compliance levels include M1, M2, M2.1, M3a, and M4a. Unless otherwise stated in this specification, each level builds upon the requirements of the previous (lower) level, with any additional requirements or exceptions documented in each level.

NOTE: M3a and M4a describes preliminary definitions of future compliance levels, for the purpose of public review and feedback. The 'a' denotes that this is an alpha work-in-progress compliance level. These definitions are subject to change in future publications of this specification.

Table 4 below shows the summary of SBMR Compliance levels.

Table 4: SBMR Compliance Levels

Level	Out-of-band Interface	SoC Side-band Interface	Host/SoC In-band Interface	BMC IO Device Interface	BMC Platform Element Interface
M0	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED
M1	Required IPMI	IMPLEMENTATION DEFINED	Required IPMI SSIF	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED
M2/ M2.1	Required Redfish and IPMI	IMPLEMENTATION DEFINED	Required IPMI SSIF and Redfish Host Interface	Conditional Requirement If shared physical NIC Interface - NC-SI is required	IMPLEMENTATION DEFINED
M3a	Required Redfish	Required MCTP over I2C/SMBUS	Required IPMI SSIF and either Redfish Host Interface or MCTP (Physical Interface TBD)	Conditional Requirement If shared physical NIC is used NC-SI over RBT or MCTP	Refer to [15] and [8] for guidance
M4a	Required Redfish	Required MCTP over I3C or MCTP (Physical Interface TBD)	Required Redfish Host Interface and MCTP (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

2.1 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.

2.2 Level M1

The requirements for level M1-based servers are defined in this section, and illustrated in Figure 2 below.

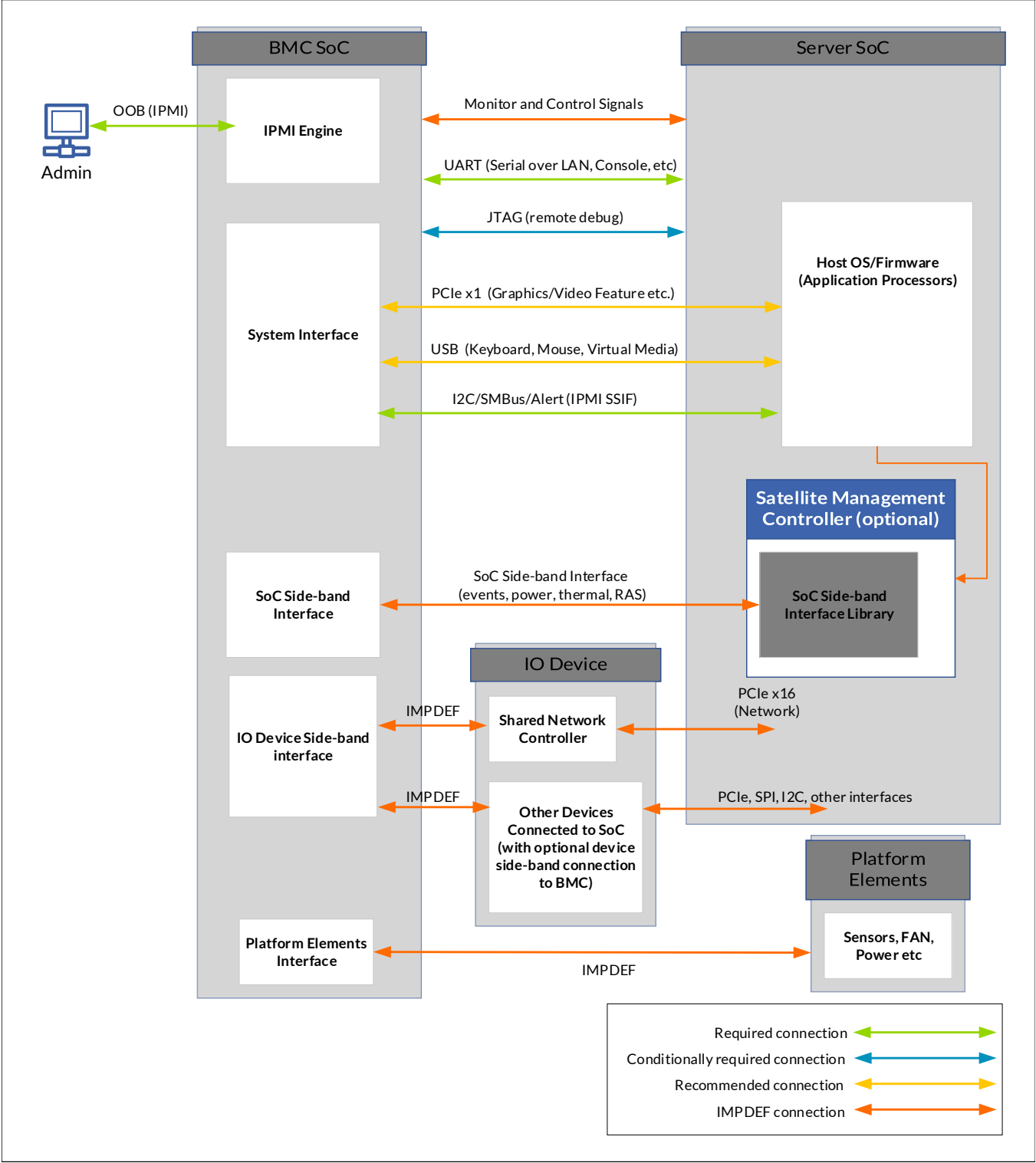


Figure 2: Server Management Interfaces (Level M1)

2.2.1 SoC-BMC interface

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.

2.2.1.1 Requirements

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 [8] defines four supported physical and logical interfaces, including 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. The recommended SMBus slave address is 20h, as stated by the IPMI specification. However, this is just a recommendation, and the actual value used is platform specific, and must match whatever value that is hardcoded in the platform firmware or in the Arm SoC.

Console UART

The Arm SoC must have at least one SBSA [1] compliant UART connection to the BMC for the purpose of serial-over-LAN (SoL) support. This is required for the Host Software, such as OS or UEFI, console input/output redirection.

Per the SBSA [1] and SBBR [16], the console UART must be an SBSA [1] compliant UART that must be exposed to the host software using the Serial Port Console Redirection (SPCR) ACPI [5] Table. Default baud rate for interoperability with commercially available BMCs is required to be 115200 bits/second.

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

2.2.1.2 Recommendations

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 video redirection.

USB

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

JTAG

Remote Debug is an invasive or non-invasive external debug, through a physical interface, such as 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.
- In a multi-socket system, 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) [17]

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 [18] or ADiv6 [19].

- 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 Section E.

2.2.2 BMC-platform elements interface recommendations

The BMC-Platform Elements interface for Level-M1 based server systems is IMPLEMENTATION DEFINED. Typically, the SMBus/I2C medium is used.

2.2.3 BMC management services (out-of-band) interface recommendation

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

2.3 Level M2

The requirements for level M2-based servers are defined in this section, and illustrated in Figure 3 below.

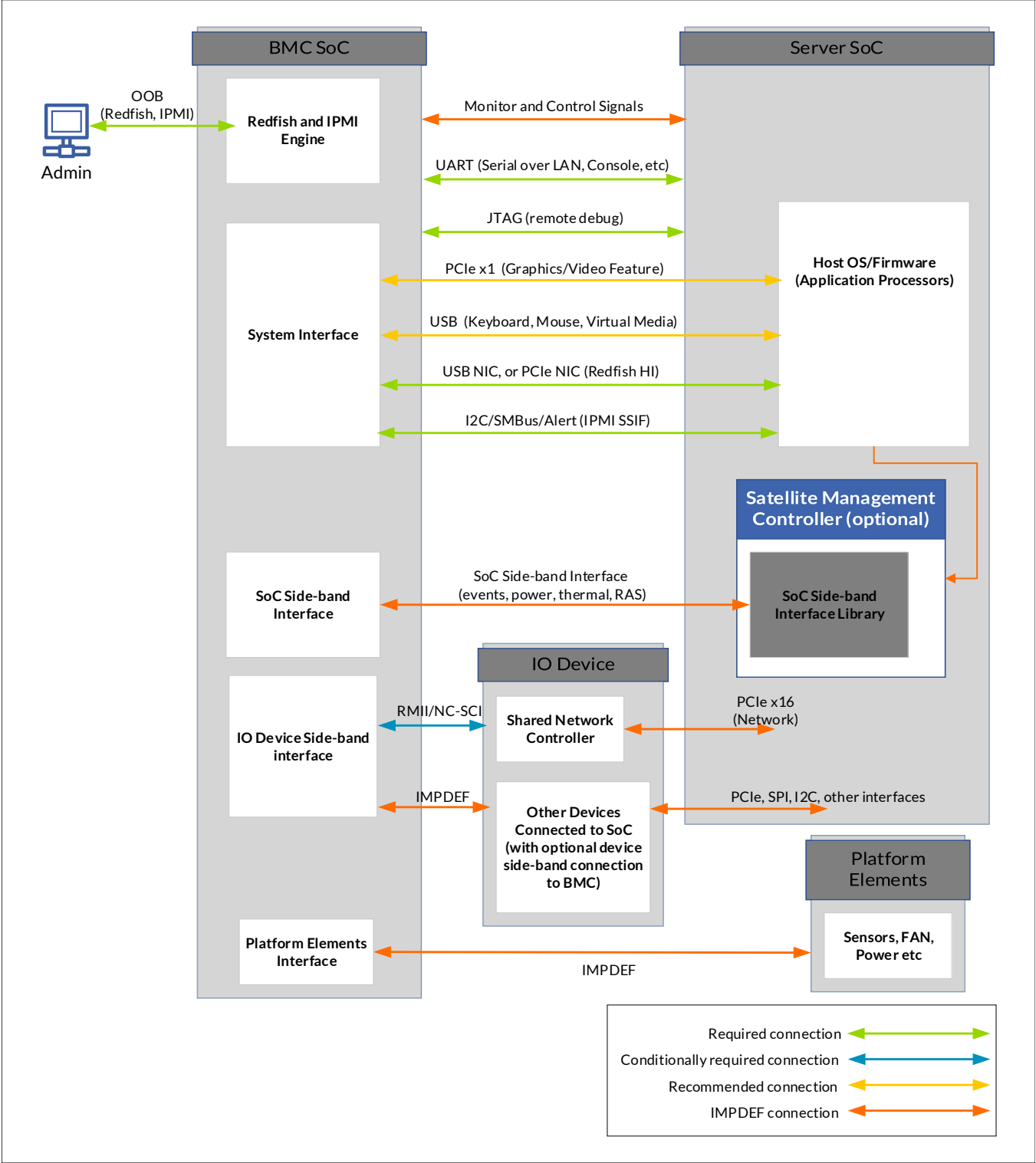


Figure 3: Server Management Interface (Level M2)

2.3.1 SoC-BMC interfaces

2.3.1.1 Requirements

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

Host SoC in-band interface

The Host/SoC In-Band interface must be compliant to the Redfish Host Interface Specification [13]. The Arm SoC must expose this interface using one of the following physical interfaces:

- 1) The Arm SoC must have a USB connection, using either on-chip USB support or external onboard USB support with a PCIe USB device, to the BMC. This is required for Redfish Host Interface 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. This is required for Redfish Host Interface communication over PCIe network device.

NOTE: In addition to USB or PCIe network device, the Redfish Host Interface Specification [13] defines an OEM proprietary method. This proprietary method is not recommended for M2-compliant systems.

In addition to the Redfish Host Interface, M2-compliance requires that a second Host-SoC in-band interface based on IPMI must exist.

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.

2.3.1.2 Recommendations

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

2.3.2 BMC-platform elements interface

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

2.3.3 BMC-IO device interface

When using a shared physical NIC interface between the BMC and the Arm SoC, then Network Controller Sideband Interface (NC-SI)[20] over reduced media independent interface (RMII) based transport is required for Level M2-based server systems.

NC-SI[20] defines a combination of logical and physical paths that interconnect the BMC and Network Controller(s) for the purpose of transferring management communication traffic. NC-SI includes the 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 Level M2-based server systems is IMPLEMENTATION DEFINED.

2.3.4 BMC management services (out-of-band) interface

Level M2-based server systems requires that the BMC management services interface supports the Redfish Interface [2] .

IPMI support is also a requirement for M2-compliant server systems. Refer to Section B.1 for the minimal IPMI commands required.

Level M2-based server systems further standardize the BMC management services interface by adopting the Redfish Interoperability Profiles Specification [21] and the individual profiles contained in the Redfish Interoperability Profiles Bundle [22] .

Supporting OpenCompute Project (OCP) defined profiles is required for OCP compliant servers. OCP currently defines two Redfish profiles for hardware management:

1. OCP Baseline Hardware Management Redfish Profile [9] . This is the minimum level a Redfish interface must provide for OCP compliant hardware management.
2. OCP Server Hardware Management Redfish Profile [10]. This profile defines additional requirements on top of the OCP Baseline profile [9] for OCP compliant server hardware management.

As Redfish Schema [7] definitions are designed to provide significant flexibility and allow conforming implementations on a wide variety of products, few properties within the Redfish Schemas are required. 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 [9] [10]).

2.4 Level M2.1

The requirements for level M2.1-based servers are defined in this section, and illustrated in Figure 4 below.

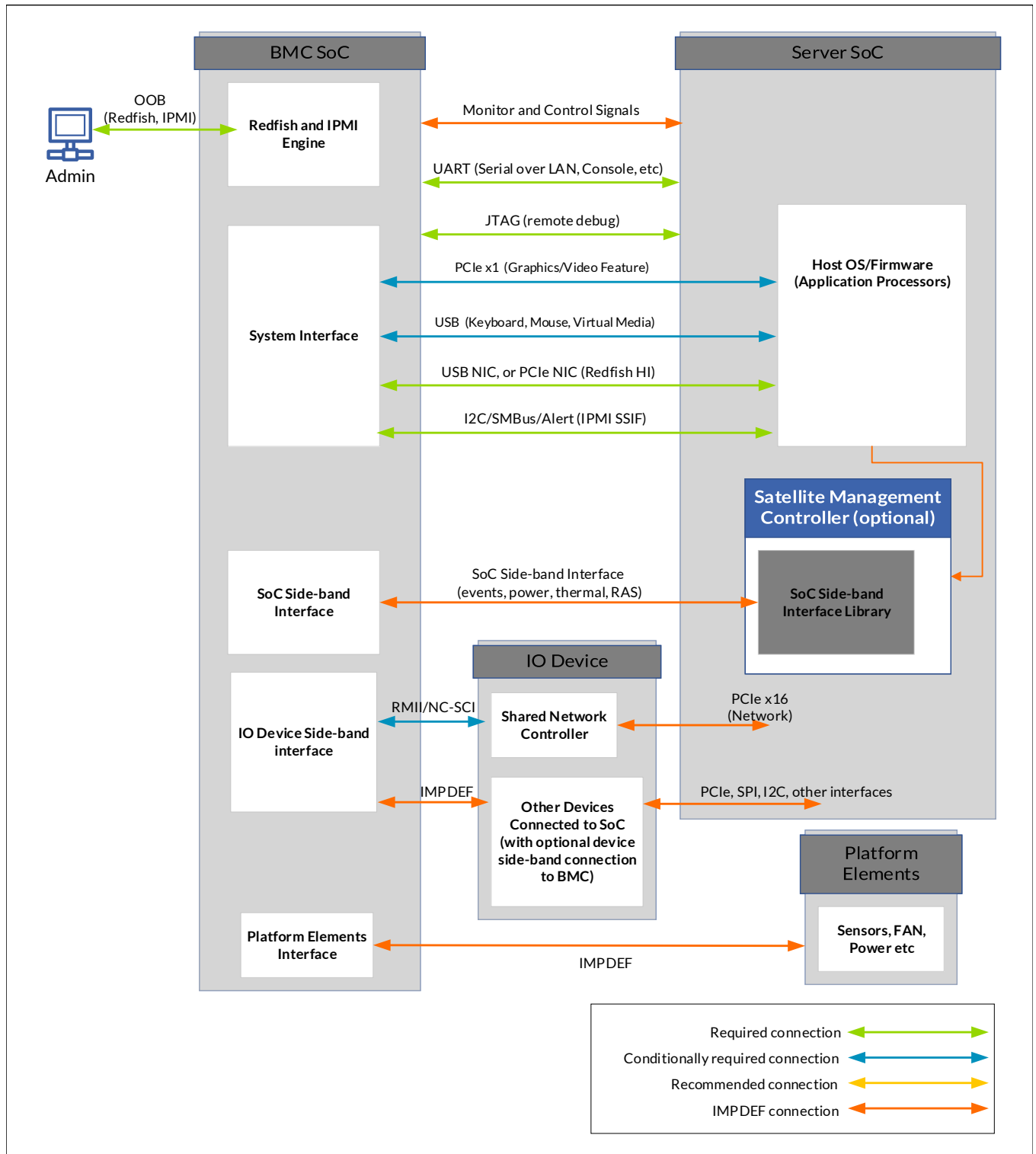


Figure 4: Server Management Interface (Level M2.1)

2.4.1 SoC-BMC interfaces

2.4.1.1 Requirements

The requirements for these interfaces on Level M2.1 based server systems are the same as the requirements for Level M2 based server systems, with some additional requirements.

Host SoC in-band interface

The In-Band SSIF interface must follow the IPMI specification clarifications that are outlined in Appendix Section B.3 and Section B.4 of this specification. The SSIF interface must also support an SMBAlert pin to enable BMC notification to the host and improve the performance of the In-Band SSIF interface communication. The recommended SMBus slave address is 20h, as stated by the IPMI specification. The actual value that is used is platform specific, and must match whatever value that is hardcoded in the platform firmware or in the Arm SoC.

Level M2.1-based server systems must implement the following IPMI commands:

- Industry standard commands
 - Remote Power Control Section B.1.1
 - Boot Device Selection Section B.1.2
 - BMC/Host Mapping Section B.1.3
 - BMC User Manipulation Section B.1.4
 - Redfish Host Interface Bootstrapping Section B.1.5. This is required only if the platform supports bootstrapping Redfish Host Interface temporary credentials to the OS.
- Arm-defined IPMI commands
 - Send Platform Error Record Section C.2.2.1. This is required only if the platform supports reporting platform errors to the BMC using the in-band interface.
 - Send Boot Progress Code Section F. This is required only if the platform supports reporting boot progress codes to the BMC.

PCIe

In level M1, the PCIe connection between the BMC and the SoC is a recommendation. In level M2.1, the interface is upgraded to a conditional requirement in systems that support remote Keyboard-Video-Mouse (KVM). This interface is not required to support legacy VGA functionality.

USB

In level M1, the USB connection between the Arm SoC and the BMC is a recommendation. In level M2.1, the interface is upgraded to a conditional requirement in systems that support remote Virtual Media or KVM.

2.4.1.2 Recommendations

The recommendation for the SoC-BMC interfaces for Level M2.1-based server systems are the same requirements and recommendations as for Level M2-based server systems.

2.4.2 BMC-platform elements interface

The BMC-Platform Elements interface requirements and recommendations for Level M2.1-based server systems are the same requirements and recommendations as for Level M2-based server systems.

2.4.3 BMC-IO device interface

The requirements and recommendations for the BMC-IO device interfaces for Level M2.1-based server systems are the same requirements and recommendations as for Level M2-based server systems.

2.4.4 BMC management services (out-of-band) interface

The requirements and recommendations for the BMC Out-of-band interfaces for Level M2.1-based server systems are the same requirements and recommendations as for Level M2-based server systems.

2.5 Level M3 alpha (work-in-progress)

The requirements for level M3-based servers are defined in this section, and illustrated in Figure 5 below.

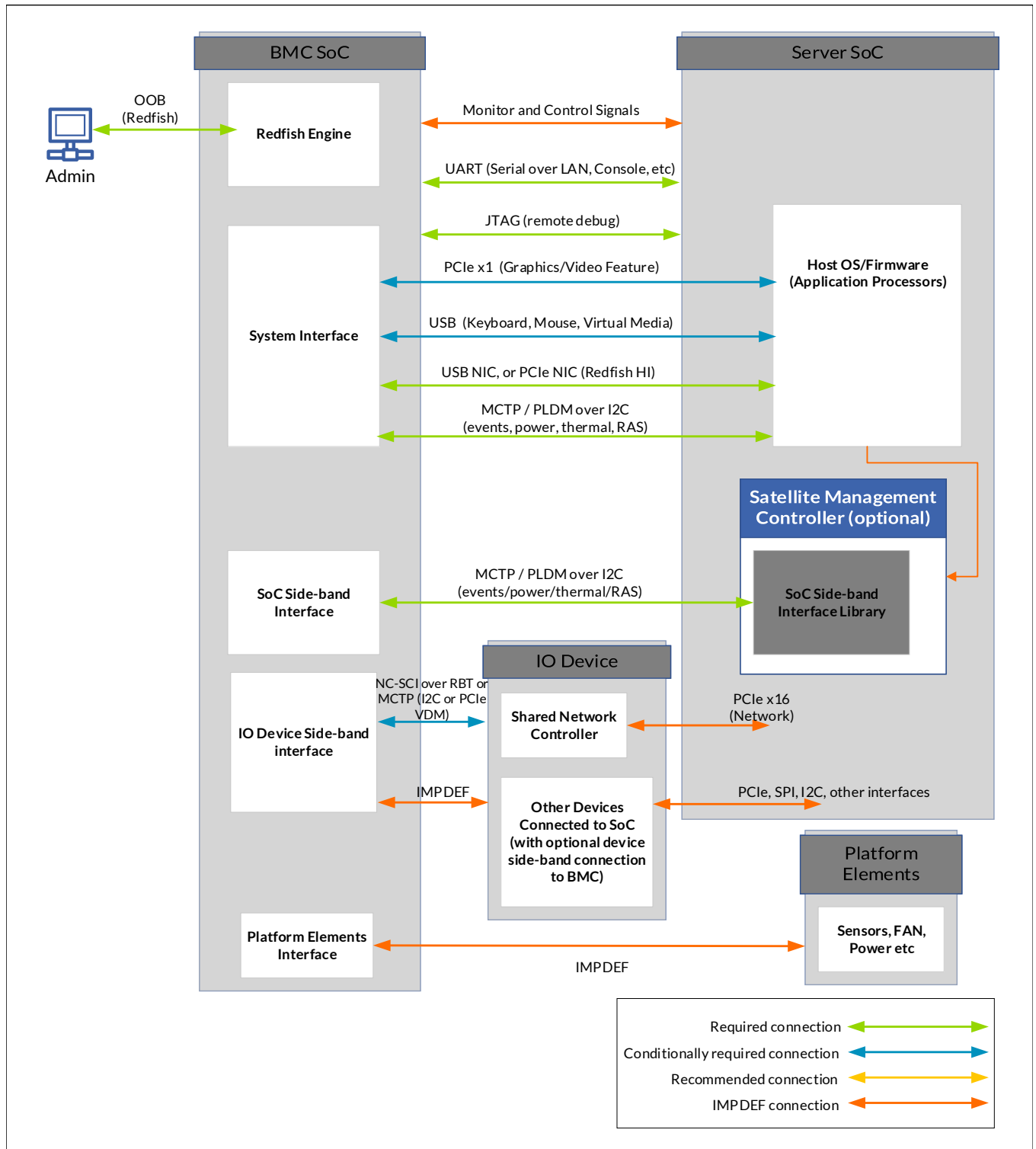


Figure 5: Server Management Interface (Level M3a)

2.5.1 Requirements

The requirements for these interfaces on Level M3 based server systems are the same as the requirements for Level M2.1 based server systems, with some additional requirements.

2.5.2 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 [3][23][24][25][26] 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 [4][27] 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 [28] 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 [29].

2.5.3 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 [8].

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

2.5.4 BMC-IO device interface recommendations

If using shared physical NIC interface between BMC and SoC, then Network Controller Sideband Interface (NC-SI)[20] over reduced media independent interface (RMII) based transport or MCTP is required for Level M3 based server systems. If NC-SI over MCTP [30] is used, the physical layer used is one of the transport bindings on which MCTP can be bound (for example, PCIe VDM [14] or SMBus/I2C).

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

2.6 Level M4 alpha (work-in-progress)

The requirements for level M4-based servers are defined in this section, and illustrated in Figure 6 below.

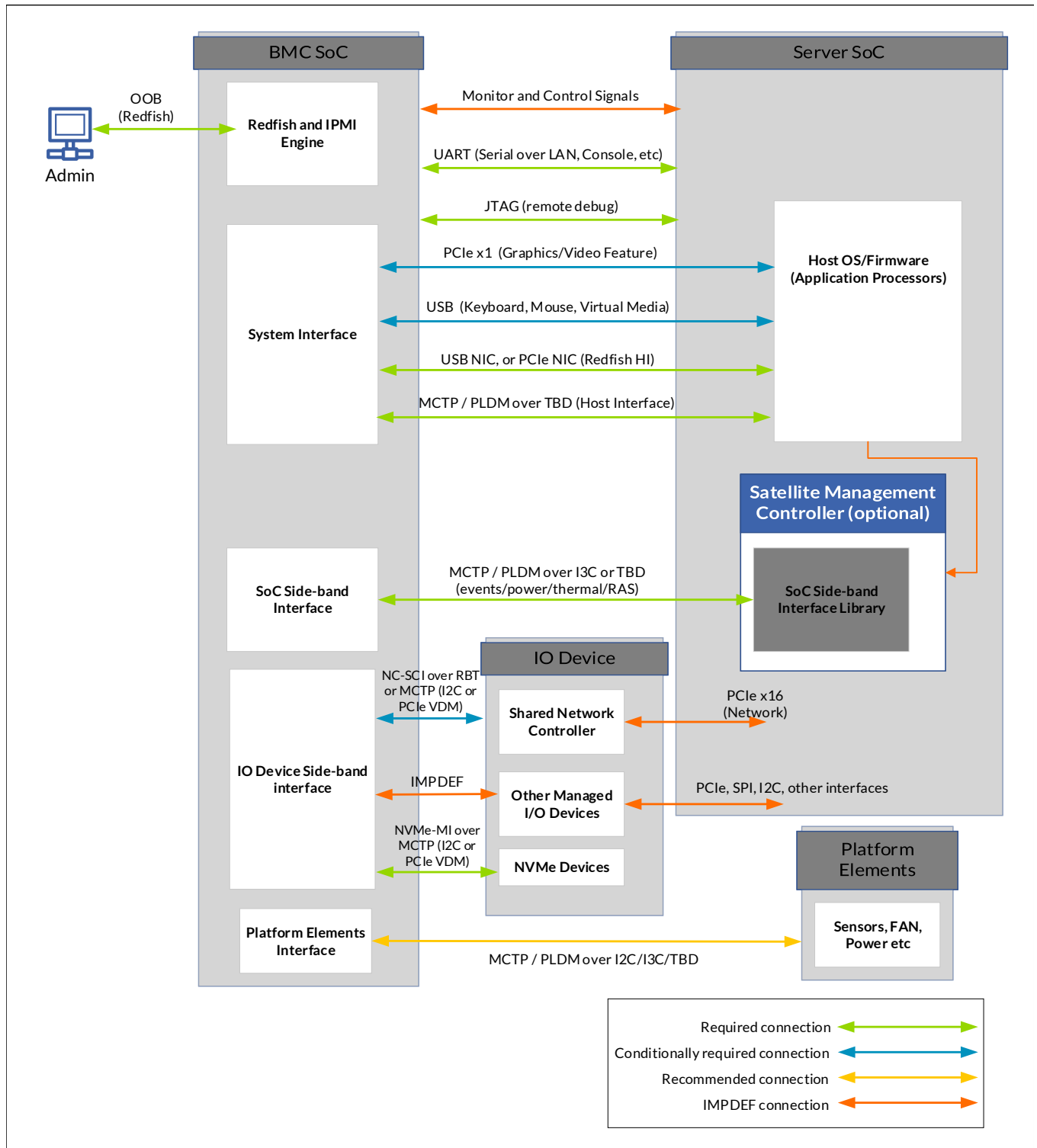


Figure 6: Server Management Interface (Level M4a)

2.6.1 Requirements

The requirements for these interfaces on Level M4 based server systems are the same as the requirements for Level M4 based server systems, with some additional requirements.

2.6.2 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)

2.6.3 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[3][23][24][26] 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 [4][27] 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 [28] as the format of PLDM over MCTP messages.
- PLDM for Redfish Device Enablement [31] 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/I2C medium, the physical and data-link layer methods for MCTP communication are defined by the MCTP over SMBus/I2C binding specification [29].

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

2.6.4 BMC-IO device interface recommendations

If using shared physical NIC interface between BMC and SoC, the requirements for these interfaces on Level M4 based server systems are the same as the requirements for 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 [32] 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 Level M4 based server systems is IMPLEMENTATION DEFINED.

A OPENBMC

The OpenBMC project (<https://www.openbmc.org/>) is an open-source project that provides a Linux distribution which implements a BMC firmware stack for devices such as servers, top-of-rack switches, or storage 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 Linux Foundation project hosted at <https://github.com/openbmc/openbmc>. Facebook, Google, IBM, Intel, and Microsoft are the founding TSC members. Arm has also joined as a TSC member.

OpenBMC is a sample implementation of the BMC software. Actual deployment of BMC in SBSA[1] compliant AArch64 servers can choose to use this implementation or other commercial solutions.

B IPMI IMPLEMENTATION GUIDE

This appendix documents the minimum IPMI commands required by SBMR. It also documents the Arm specific IPMI commands that are defined by SBMR.

B.1 Standard IPMI commands

The following are IPMI commands defined by the standard IPMI specification [8] that are the required by SBMR.

B.1.1 Remote power control

B.1.1.1 Power on

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

B.1.1.2 Power off

A platform must provide a mechanism for remotely powering an individual node off. This mechanism should be provided out-of-band, 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.1.1.3 Graceful power off

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

B.1.1.4 IPMI commands required

IPMI Chassis Control Command (IPMI § 28.3)

B.1.2 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.1.2.1 IPMI commands required

The following IPMI boot device selection commands are required:

- IPMI Set System Boot Options Command (IPMI § 28.12)
- IPMI Get System Boot Options Command (IPMI § 28.13)

B.1.3 BMC to Host mapping

It should be possible to automatically determine the mapping between a host and its BMC. The host must be able to identify its BMC configuration through an in-band mechanism. Alternatively, the BMC must be able to provide unique identification information about the host, for example host MAC addresses.

B.1.4 BMC user manipulation

When an IPMI LAN capable BMC is used to provide platform interfaces, the deployment server must be able to authenticate to the BMC by using the IPMI System Interface through the in-band interface. This is required for deployment server to be able to add a private user to the BMC using 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 administrator can authenticate to the BMC over the IPMI LAN interface. This requires an IPMI-compliant BMC system Interface.

B.1.5 Redfish host interface credentials bootstrapping

The Redfish in-band Host Interface includes an optional feature to bootstrap temporary Redfish service host accounts using some IPMI commands. These commands are defined in version 1.30 or newer of the Redfish Host Interface Specification [13].

B.1.5.1 IPMI commands

- IPMI Get Manager Certificate Fingerprint Command (Redfish Host Interface § 9.1.1)
- IPMI Get Bootstrap Account Credentials (Redfish Host Interface § 9.1.2)

B.1.5.2 Redfish properties

- `CredentialBootstrapping` property defined in the `HostInterface` Redfish Schema [7] [33]. Platforms should implement this property as a writeable configuration setting to allow the administrator to disable the bootstrapping facility for security reasons.
- `CredentialBootstrappingRole` property in the `Links` property defined in the `HostInterface` Redfish Schema [7] [33].

B.1.6 IPMI support verification

This script verifies the basic remote IPMI functionality:

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

B.2 Arm standard IPMI commands

This section lists Arm standard IPMI commands that are defined by SBMR.

B.2.1 General IPMI commands format

The common components of IPMI message as defined by the IPMI specification [8] 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.
- **Group Extensions (2Ch, 2Dh):** This will allow all the commands to come under a Group for Non-IPMI groups and requests.

- SBMR uses the Group Extension NetFn (2Ch, 2Dh) option from the IPMI specification [8]. This is because it gives the Arm ecosystem a broad scope for managing the transport and protocols.
- **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. The first data byte position in requests, and the second byte in responses, under the Group Extension NetFn 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.
 - SBMR defines the value **AEh** as the defining body code. This value will be used for all IPMI commands defined in SBMR.

B.2.2 List of Arm standard IPMI commands

Command	NetFn	Command Code	Definition
Send Platform Error Record	2Ch	01h	Section C.2.2.1
Send Boot Progress Code	2Ch	02h	Section F

B.3 IPMI specification clarifications and corrections

The following section lists corrections and clarifications to the IPMI specification [8] that directly impact IPMI implementation on Arm-based SBMR systems, including complete support for IPMI SSIF interface. These corrections are listed here rather than the official IPMI Specification because “No further updates to the IPMI specification are planned or should be expected” by the IPMI Promoters group.

IPMI §	Existing language	Updated language
12	“SSIF encapsulates IPMI messages and transfers them between the host controller and BMC using the SMBus “Write Block” and “Read Block” protocols. With SSIF, the BMC is always accessed as a slave device on SMBus. The host controller masters the to write data to the BMC. ”	“SSIF encapsulates IPMI messages and transfers them between the host controller and BMC using the SMBus “Write Block” and “Read Block” protocols. With SSIF, the BMC is always accessed as a slave device on SMBus. The host controller masters the write data to the BMC. ”
12.3	“The combination of a Start transaction followed by an End transaction can transfer up to 63 bytes of IPMI message. The Middle transaction is available when there is a need to transfer an IPMI message of greater than 64 bytes. As of this writing, there are no standard IPMI messages to the BMC that are longer than 63 bytes . Therefore, the ‘middle’ transaction is defined solely as needed by any OEM/group network functions (network function codes 2Ch:3Fh) in the particular BMC Implementation”	“The combination of a Start transaction followed by an End transaction can transfer up to 64 bytes of IPMI message. The Middle transaction is available when there is a need to transfer an IPMI message of greater than 64 bytes. As of this writing, there are no standard IPMI messages to the BMC that are longer than 64 bytes . Therefore, the ‘middle’ transaction is defined solely as needed by any OEM/group network functions (network function codes 2Ch:3Fh) in the particular BMC Implementation”

IPMI §	Existing language	Updated language
12.3.1	Table 12 - BMC Multi Part End	Table 12 - BMC Multi Part End (see below)

Table 12

Existing						Correction					
Slave address (7)	R/W=0 (1)	SMBus CMD =07h	Length	IPMI Data	[PEC]	Slave address (7)	R/W=0 (1)	SMBus CMD =08h	Length	IPMI Data	[PEC]

B.4 SSIF single and multi-part transactions

The SMBus System Interface (SSIF) defines two types of writes, a single-part write, and a multi-part write. Multi-Part writes are used when more than 32-bytes of IPMI message data need to be written to the BMC. For any IPMI commands, where the data size is greater than 32 bytes, SBMR recommends the use of multi-part writes.

A multi-part write has one Start (SMBus CMD=0x06), zero or more Middle (SMBus CMD=0x07), and one End (SMBus CMD=0x08) transactions.

Multi-part Start transaction looks like this:

Byte 1		Byte 2	Byte 3	Byte 4		Byte 5	Byte 6+	
Slave Address (7 bits)	R/W (1 bit)	SMBus CMD	Length (8 bits)	NetFN (6 bits)	LUN (2 bits)	IPMI CMD (8 bits)	data (1 or more bytes)	[PEC] (8 bits)
	0	0x06	0x20	0x2c		0x##	0xAE Followed by Data bytes	

Note that the NetFun code is “0x2C” and the first byte of IPMI request data is “0xAE” to indicate that the IPMI commands are defined by SBMR.

Multi-part Middle transactions look like this:

Byte 1		Byte 2	Byte 3	Byte 4+	
Slave Address (7 bits)	R/W (1 bit)	SMBus CMD	Length (8 bits)	data (1 or more bytes)	[PEC] (8 bits)
	0	0x07	0x20	Followed by Data bytes	

Multi-part End transactions look like this:

Byte 1		Byte 2	Byte 3	Byte 4+	
Slave Address (7 bits)	R/W (1 bit)	SMBus CMD	Length (8 bits)	data (1 or more bytes)	[PEC] (8 bits)
	0	0x08	<= 0x20	Followed by Data bytes	

Considering the clarifications of the IPMI Specification in Section B.3, the following are some examples of multi-write SSIF transactions of different sizes:

Example 1: sending <= 32 bytes:

- 1st Write transaction: SMBus = 0x6, Length = 0x20

Example 2: sending 64 bytes:

- 1st Write transaction: SMBus = 0x6, Length = 0x20
- 2nd Write transaction: SMBus = 0x8, Length = 0x20

Example 3: sending 95 bytes:

- 1st Write transaction: SMBus = 0x6, Length = 0x20
- 2nd Write transaction: SMBus = 0x7, Length = 0x20
- 3rd Write transaction: SMBus = 0x8, Length = 0x1F

Example 4: Sending 96 bytes:

- 1st Write transaction: SMBus = 0x6, Length = 0x20
- 2nd Write transaction: SMBus = 0x7, Length = 0x20
- 3rd Write transaction: SMBus = 0x8, Length = 0x20

C RAS MESSAGE FORMAT

C.1 Level M0

For Level M0-based server systems, the transfer of RAS error records over in-band, side-band, and out-of-band interfaces is IMPLEMENTATION DEFINED.

C.2 Level M1

Figure ?? shows a conceptual illustration of required IPMI based in-band, SoC side-band, and out-of-band RAS interfaces for Level M1-based server systems.

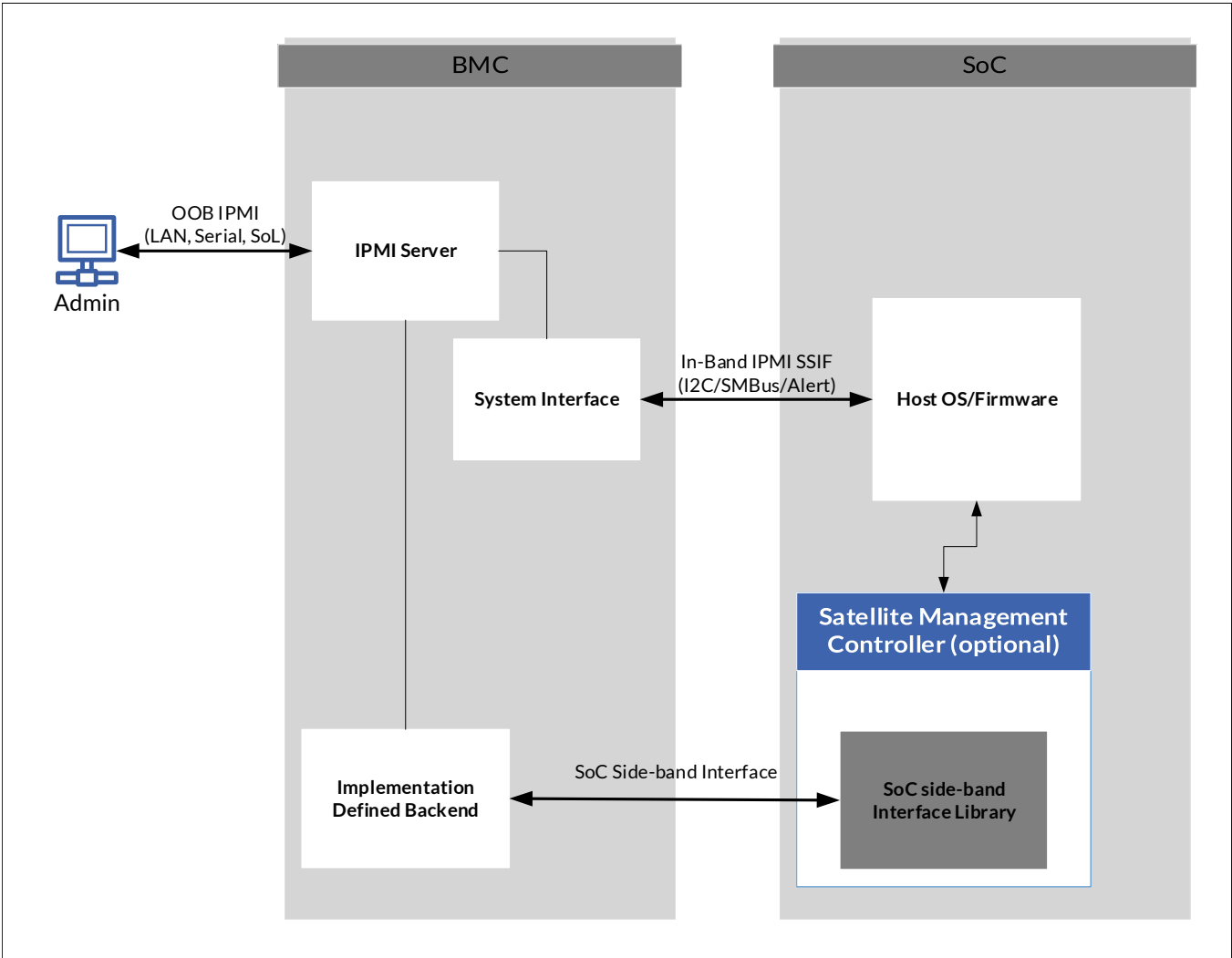


Figure 7: IPMI based RAS Interfaces

C.2.1 SMBus System Interface (SSIF) in-band interface

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

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

Figure 8 illustrates the overview of RAS Events interaction with the event receiver and RAS Manager through SMBus System Interface (SSIF).

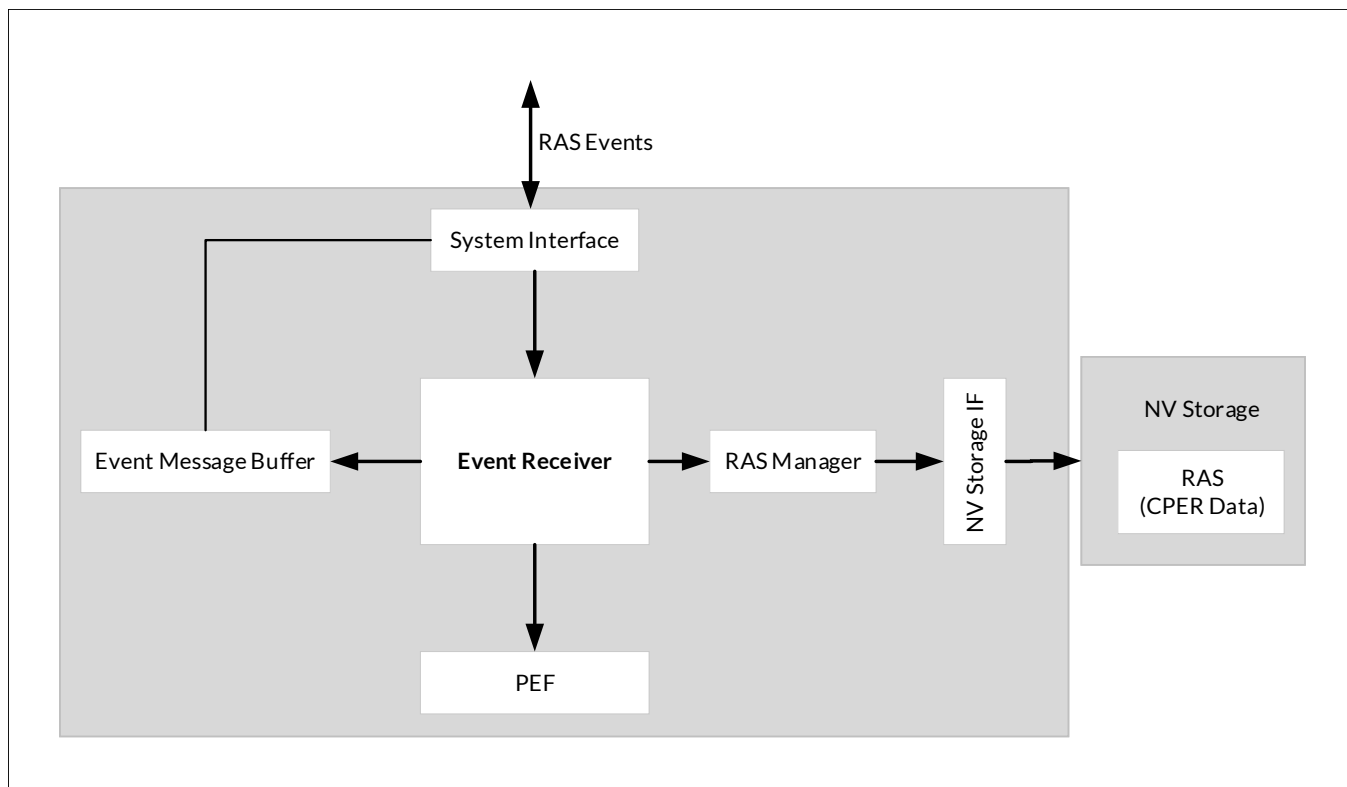


Figure 8: IPMI based RAS Event Receiver

Figure 8 represents a conceptual illustration of the way that 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 the Host OS or host firmware.

When the BMC receives a message via the system interfaces, a BMC firmware 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 (CPEP) entry. Finally, the RAS Manager function calls the Non-Volatile Storage Interface to store the event record.

SBMR recommends the error record data format to be in raw Common Platform Error Record (CPEP) format when using this interface. The format of CPEP is defined in UEFI Specification [6] (UEFI § N). When creating CPEP raw files for logging to, or extracting from, the BMC, SBMR recommends that the file is limited to the specific CPEP error section type, as defined by (UEFI § N.2.2), and not include the CPEP Record Header from (UEFI § N.2.1). This applies to all methods described in this appendix: IPMI, Redfish, and MCTP.

C.2.2 RAS IPMI message format

The RAS (CPER) IPMI commands follow the general format of Arm defined IPMI commands as outlined in Section B.2, with Group Extension 2Ch, and defining body AEh.

C.2.2.1 Send Platform Error Record (NetFn 2Ch, Command 01h)

This command is used to send the RAS CPER error record to the BMC.

Request Data

Bytes	Data field
1	Group extension defining body (AEh)
2...n	CPER Error record

Response Data

Bytes	Data field
1	Group extension defining body (AEh)
2...n	CPER Error record

Because the size of RAS CPER error record format is in the order of KBs, SBMR recommends the use of SSIF multi-part write transaction. For information on SSIF multi-part transactions, refer to Section B.3.

C.2.3 SoC side-band interface

RAS error records can be generated in the host OS or the firmware, then transferred over to the Satellite Management Controller (SatMC). RAS error records can also be generated in the SatMC itself. For both cases, the transport of these error records over the SoC Side-band interface is IMPLEMENTATION DEFINED for SBMR Level-M1 compliant server systems.

C.2.4 Out-of-band interface

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

C.3 Level M2 and Level M2.1

Level M2 and Level M2.1 require Redfish as the out-of-band interface, and both Redfish and IPMI Host Interfaces as the in-band interfaces. Figure 9 shows a conceptual illustration of these interfaces for RAS.

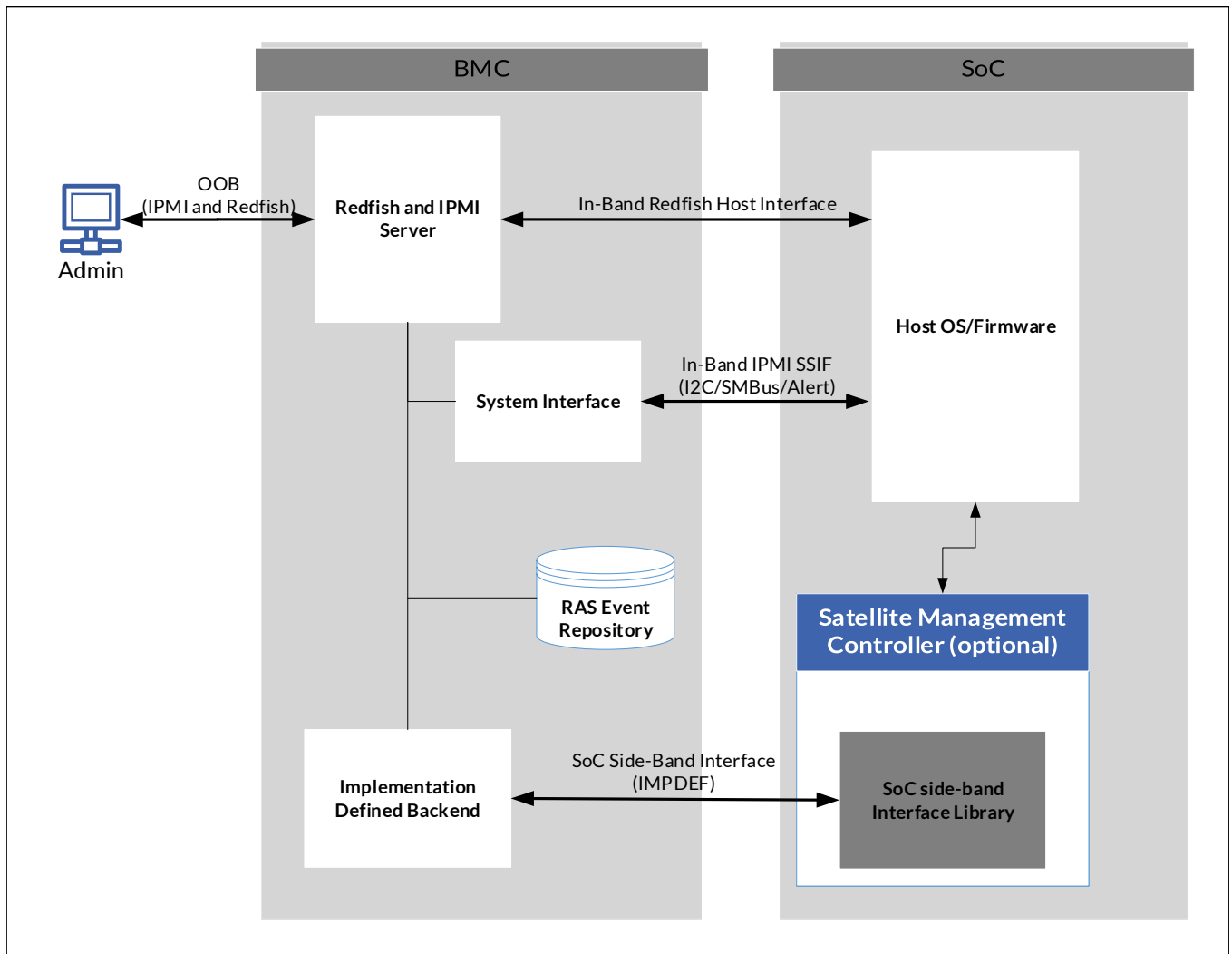


Figure 9: Redfish and IPMI based RAS Interfaces

C.3.1 Redfish and IPMI host (in-band) interfaces

For transferring RAS error records generated in Host OS/Firmware to the BMC, SBMR recommends IPMI System Interface as the in-band interface for the Levels M2 and M2.1 based server systems, as defined in Section C.2 for Level-M1 server systems.

Arm recommends storing the error records 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

The Redfish model for extracting Platform Error Records is defined in DMTF Redfish Schema Supplement [42] version 2020.3, under the LogEntry v 1.7.0 schema. A LogEntry object may contain the following properties to point to the platform error record diagnostic binary blob:

- `AdditionalDataURI`: Pointer to the platform error record binary file that can be downloaded by a client. SBMR recommends that this file to be formatted as a CPER binary raw file.
- `AdditionalDataSizeBytes`: Size of the diagnostics binary file in bytes
- `DiagnosticDataType`: The type of the diagnostics binary file. For platform error records, this can be `OS`, `PreOS`, or `OEM`.
 - SBMR recommends setting this property to `OEM`, and setting the `OEMDiagnosticDataType` property to `CPER`. This allows user software to distinguish CPER records from other diagnostics files. This value may be standardized in a future DMTF specification.

C.3.3 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 Levels M2-compliant and M2.1-compliant systems is IMPLEMENTATION DEFINED.

C.3.4 Out-of-band interface

SBMR recommends a Redfish-based tool to extract the stored RAS error records in CPER-like format from RAS event repository.

C.4 Level M3a and M4a

Level M3a 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. Figure 10 shows a conceptual illustration of these interfaces for RAS.

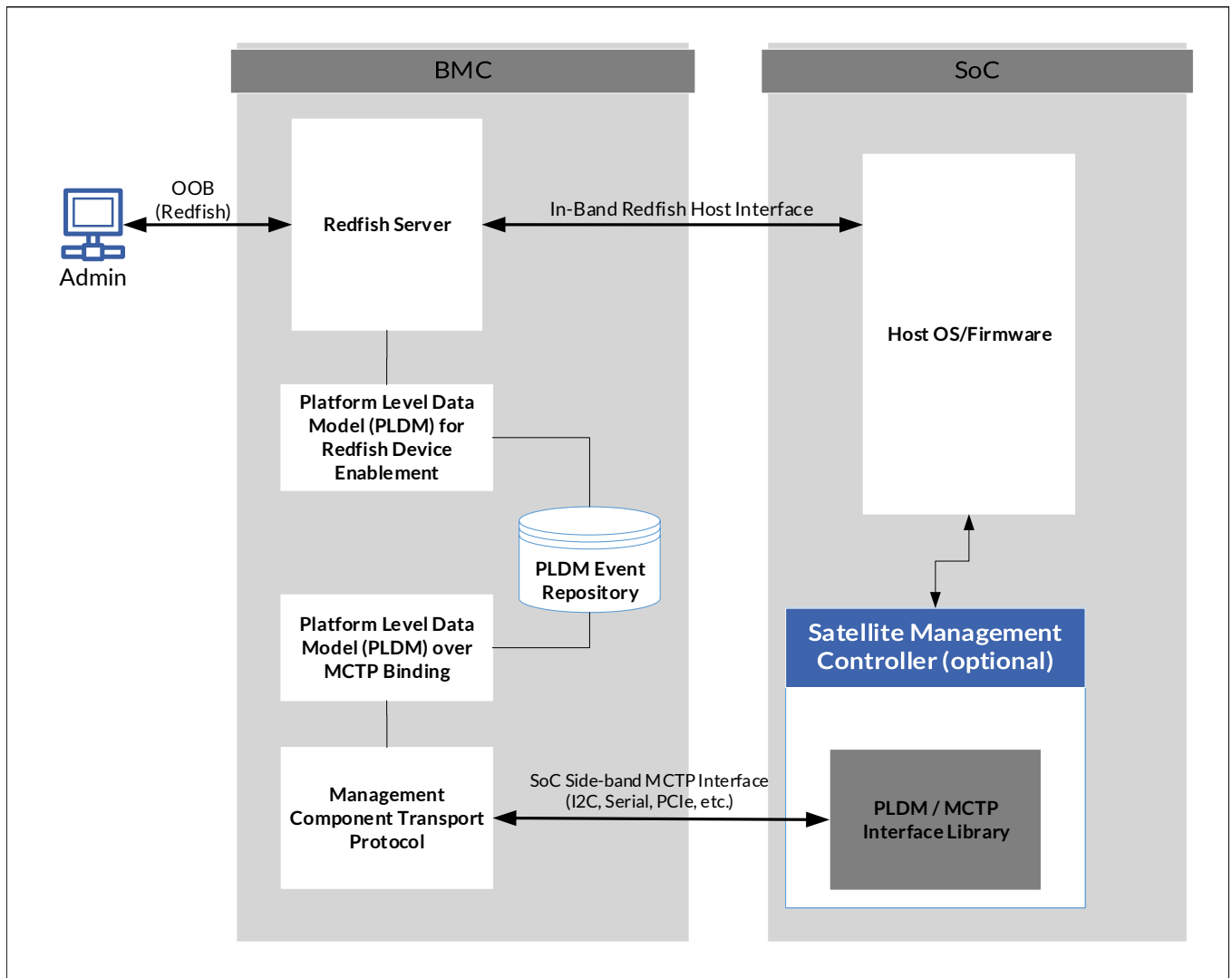


Figure 10: Redfish/PLDM/MCTP based RAS Interfaces

C.4.1 Redfish host (in-band) interface

For transferring RAS error records generated in host OS or firmware, the recommendations for Level M3-based server system are the same recommendations as for Levels M2-based and M2.1-based server systems.


C.4.2 MCTP (SoC side-band) interface

RAS error records can be generated in the host OS or the firmware, then transferred over to the Satellite Management Controller (SatMC). RAS error records can also be generated in the SatMC itself. For both cases, SBMR recommends that the transport of these error records over the SoC Side-band interface to use the Management Component Transport Protocol (MCTP) for the level M3a-based and M4a-based server systems.

The physical binding of MCTP is left best to System Implementors. Some examples of the physical binding include MCTP over I2C, MCTP over PCIe VDM, and MCTP over Serial.

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.

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.

If 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 re-assembled 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`.

For more details, refer to PLDM for Platform Monitoring and Control Specification [24]. Figure 11 shows an example flow that demonstrates switching to polled event transfer to receive an CPER event with large event data.

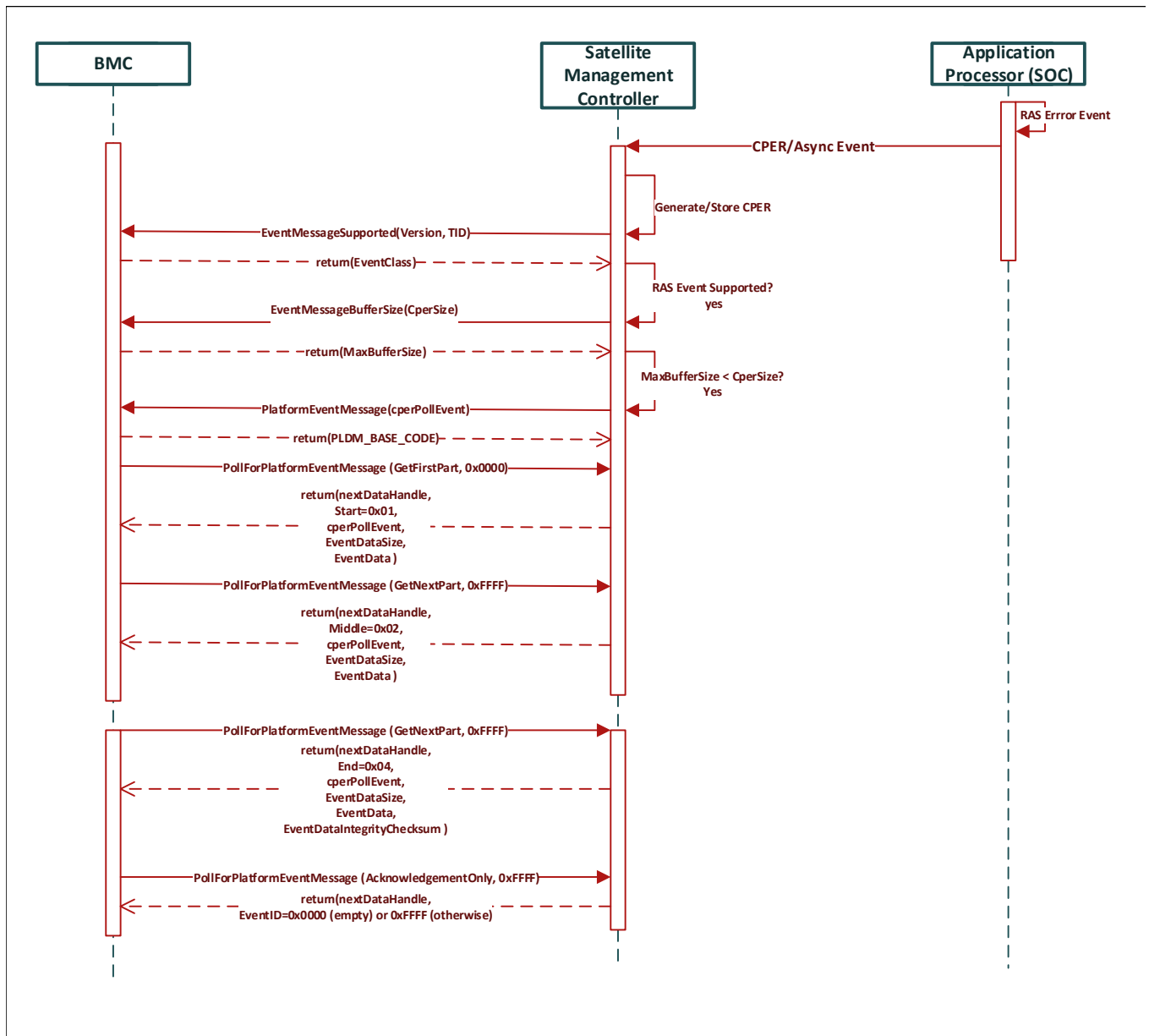


Figure 11: Redfish/PLDM/MCTP based RAS Interfaces

C.4.3 RAS PLDM message format

The proposed RAS/CPER PLDM event log entry format is shown in Table 13.

Table 13: 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.

Byte	Type	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 14.

Table 14: RAS/CPER PLDM event log entry 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 15.

Table 15: RAS/CPER PLDM event log entry 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.

Byte	Type	Field
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 16.

Table 16: RAS/CPER PLDM event log entry 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 `sectionType` field and is beyond the scope of this document.

For more details on standard and non-standard sections, refer to UEFI Specification [6] (UEFI § N).

C.4.4 Out-of-band interface

SBMR recommends a Redfish based tool to extract the stored RAS error records in a CPER-like format from the PLDM event repository.

D PLATFORM MONITORING AND CONTROL IMPLEMENTATION GUIDE

D.1 Introduction

A 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, and chassis. Examples of logical entities include virtual processors, cooling domains, and system security states.

D.2 IPMI commands to monitor and control managed entities

SBMR recommends the following list of IPMI commands for 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, refer to the IPMI Specification [8].

Sensor data records (SDRs)

SBMR recommends SDR Type 01h, Full Sensor Record, to describe the managed entities. For more details, refer to the IPMI Specification [8].

SBMR recommends the following list of IPMI commands for 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 SDR format consists of three main 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 the IPMI Specification for managed entities. For more details on the Sensor Type values, refer to the IPMI specification (IPMI § Table 42-3) [8].

For a list of sensor unit codes, refer to the IPMI Specification (IPMI § Table 43-15) [8].

Entity IDs

SBMR recommends the use of Entity IDs which identify the sensor association with a physical container. SBMR reserved the following Entity IDs to identify SoC firmware (E.g., pre-EFI firmware), SoC Management Software (E.g., Satellite/Service Management Software). These values are reserved from the OEM System Integrator defined range 0xD0 – 0xFF.

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, refer to IPMI Specification (IPMI § Table 43-13) [8].

D.3 Redfish schema to monitor and control managed entities

SBMR recommends the use of the schema for sensor as defined by DMTF [7][2]:

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 for monitoring and control of SoC-connected Numeric and State managed entities/effectors:

1. SetNumericSensorEnable
2. GetSensorReading
3. InitNumericSensor
4. SetStateSensorEnables
5. GetStateSensorReadings
6. InitStateSensor
7. SetNumericEffectorEnable
8. SetNumericEffectorValue
9. GetNumericEffectorValue
10. SetStateEffectorEnables
11. SetStateEffectorStates
12. GetStateEffectorStates

Platform Descriptor Records (PDRs)

SBMR recommends the use of Platform Descriptor Records (PDRs). PDRs provide semantic information for managed entities.

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

1. GetPDRRepositoryInfo
2. GetPDR
3. RunInitAgent

For more details on the PLDM Commands, refer to PLDM for Platform Monitoring and Control Specification [\[24\]](#).

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 Levels M1, M2, M2.1

This section describes a reference solution for implementing BMC remote debug using OpenOCD <http://openocd.org/> for SBMR Levels M1, M2, and M2.1 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 12.

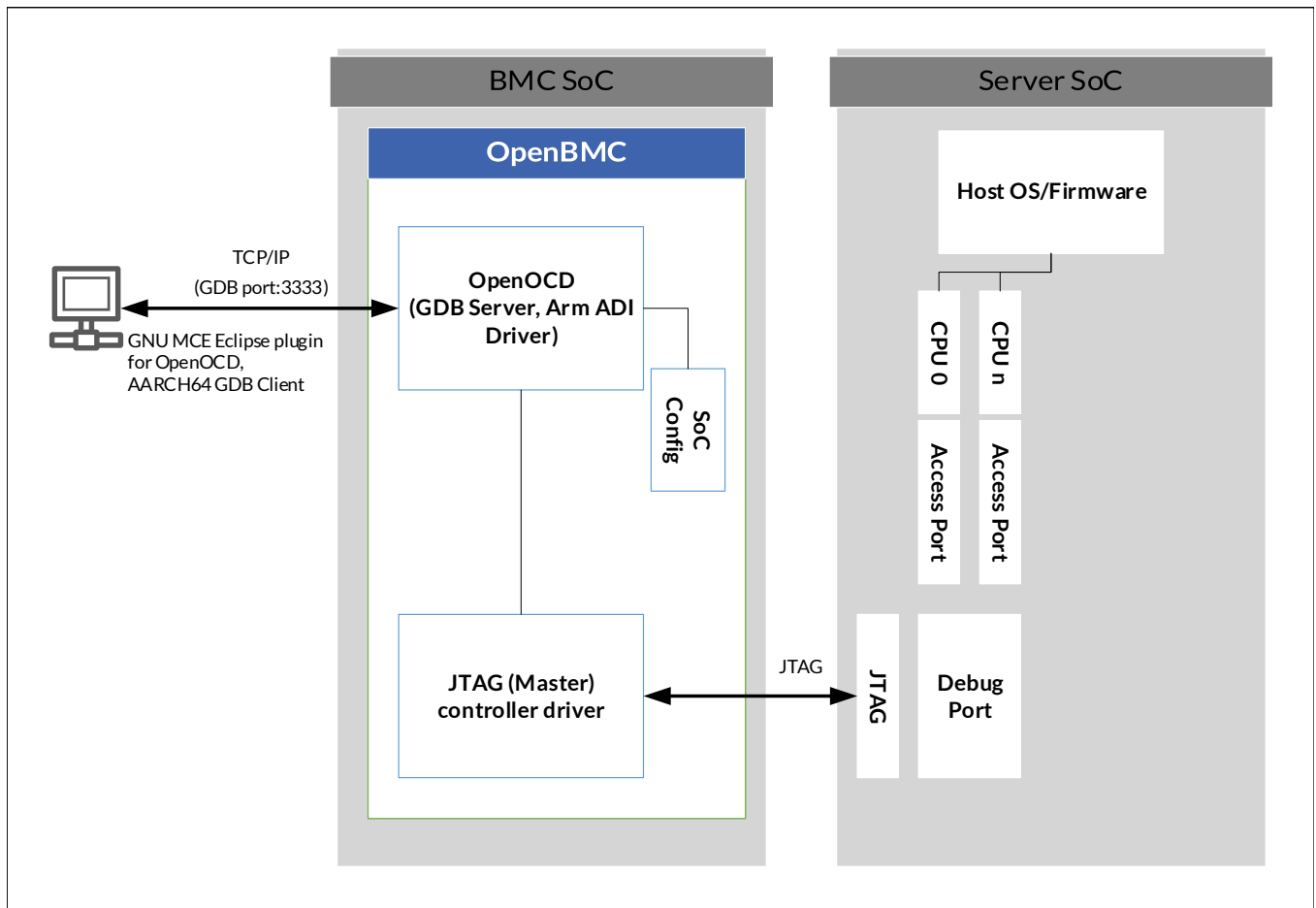


Figure 12: Reference implementation of remote debug.

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 [34].

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

F BOOT PROGRESS CODE REPORTING

F.1 IPMI commands for boot progress codes

The Boot Progress Code IPMI commands follow the general format of Arm-defined IPMI commands as outlined in Section B.2, with Group Extension 2Ch, and defining body AEh.

F.1.1 Send boot progress code (NetFn 2Ch, Command 02h)

This command is used to send the Boot Progress Code to the BMC.

Request Data

Bytes	Data field
1	Group extension defining body (AEh)
2-10	Boot Progress Code record (9 bytes). The format is defined in Section F.2 below

Response Data

Bytes	Data field
1	Completion Code: 00h: Command completed normally 80h: Command completed with error
2	Group extension defining body (AEh)

Note: Arm recommends that the caller reads the command Response Data from the BMC after sending the command “Send Boot Progress Code”. This ensures that the SSIF TX/RX buffers are emptied before sending another write.

Callers can choose to not read back Response Data after sending the command “Send Boot Progress Code”. In such cases, some SSIF transactions, especially multi-part SSIF messages, might get dropped. Whether these transactions are dropped depends on the rate in which subsequent writes are sent, and the BMC thread load. Be careful not to mix high frequency “Send Boot Progress Code” messages with multi-part SSIF messages, like the command “Send Platform Error Record”. Arm also recommends that the caller reads the response of at least the last progress code that is sent to the BMC at the end of boot.

F.1.2 Get boot progress code (NetFn 2Ch, Command 03h)

This command is used to read the last Boot Progress Code that was received by the BMC from the command “Send Boot Progress Code”.

Bytes	Data field
1	Group extension defining body (AEh)

Response Data

Bytes	Data field
1	Completion Code: 00h: Command completed normally 80h: Command completed with error
2	Group extension defining body (AEh)
3-11	Boot Progress Code record (9 bytes). The format is defined in Section F.2

F.2 Boot progress code format

The format of the data in this command follows the definitions of `EFI_STATUS_CODE_TYPE` and `EFI_STATUS_CODE_VALUE`, as defined in the PI Specification [35]. If the PI Specification adds new definitions, such as new classes, sub-classes, or operations, it is assumed that the values are valid for usage in this IPMI command.

Byte offset	Size (Bytes)	Description	Details
0	1	<code>STATUS_CODE_TYPE</code> 0x01 = <code>PROGRESS_CODE</code> 0x02 = <code>ERROR_CODE</code> 0x03 = <code>DEBUG_CODE</code>	32-bit field that follows the format of <code>EFI_STATUS_CODE_TYPE</code> as defined by the PI Specification [35] (PI § Vol 1-4.7 PI § Vol 2-14.2, PI § Vol 3- 6).
1	2	<code>STATUS_CODE_RESERVED</code> Reserved by PI Specification. set to 0x0000	
3	1	<code>STATUS_CODE_SEVERITY</code> 0x40 = <code>ERROR_MINOR</code> 0x80 = <code>ERROR_MAJOR</code> 0x90 = <code>ERROR_UNRECOVERED</code> 0xa0 = <code>ERROR_UNCONTAINED</code>	
4	2	<code>EFI_STATUS_CODE_OPERATION</code> 0x0000-0x0FFF Shared by all sub-classes in a class 0x1000-0x7FFF Subclass Specific. 0x8000-0xFFFF OEM specific. Note: This specification further divides the OEM range into the following sub-ranges: 0x8000-0xBFFF OEM/ODM reserved range 0xC000-0xDFFF SiP reserved range 0xE000-0xFFFF SBMR reserved range (for use by this specification)	32-bit field that follows the format of <code>EFI_STATUS_CODE_VALUE</code> as defined by the PI Specification [35] (PI § Vol 1-4.7 PI § Vol 2-14.2, PI § Vol 3- 6).

Byte offset	Size (Bytes)	Description	Details
6	1	EFI_STATUS_CODE_SUBCLASS Class Specific 0x00-0x7F = Defined or Reserved by PI specification 0x80-0xFF = Reserved for OEM use Note: This specification further divides the OEM range into the following sub-ranges: 0x80-0xBF OEM/ODM reserved range 0xC0-0xDF SiP reserved range 0xE0-0xFF SBMR reserved range (for use by this specification)	
7	1	EFI_STATUS_CODE_CLASS 0x00 = COMPUTING_UNIT 0x01 = PERIPHERAL 0x02 = IO_BUS 0x03 = SOFTWARE 0x04-0x7F = Reserved by the PI Specification 0x80-0xFF = Reserved for OEM use Note: This specification further divides the OEM range into the following sub-ranges: 0x80-0xBF OEM/ODM reserved range 0xC0-0xDF SiP reserved range 0xE0-0xFF SBMR reserved range (for use by this specification)	
8	1	Instance The enumeration of a hardware or software entity within the system. A system may contain multiple entities that match a class/subclass pairing. The instance differentiates between them. An instance of 0 indicates that instance information is unavailable, not meaningful, or not relevant. Valid instance numbers start with 1.	Matches the Instance parameter of ReportStatusCode() PEI service and DXE Protocol interface, as defined by the PI Specification [35] (PI § Vol 1-4.7 PI § Vol 2-14.2, PI § Vol 3- 6).

F.2.1 Example progress codes (IPMI)

The following are some examples of Boot Progress Codes that are based on standard Status Code values that are defined by the PI Specification.

Example 1 - Host processor power-on initialization

UEFI Definition	EFI_STATUS_CODE_TYPE	EFI_PROGRESS_CODE	0x00000001
	EFI_STATUS_CODE_VALUE	EFI_COMPUTING_UNIT_HOST_PROCESSOR EFI_CU_HP_PC_POWER_ON_INIT = (EFI_COMPUTING_UNIT 0x00010000) (EFI_SUBCLASS_SPECIFIC 0x00000000) = (0x00000000 0x00010000) (0x1000 0x00000000)	0x00011000

Instance	0	0x00
----------	---	------

IPMI RAW COMMAND 0x2C 0x02 0xAE 0x01 0x00 0x00 0x00 0x00 0x10 0x01 0x00 0x00

Example 2 - ResetSystem() PEI service is called

UEFI Definition	EFI_STATUS_CODE_TYPE	EFI_PROGRESS_CODE	0x00000001
	EFI_STATUS_CODE_VALUE	EFI_SOFTWARE_PEI_SERVICE EFI_SW_PS_PC_RESET_SYSTEM = (EFI_SOFTWARE 0x000F0000) (EFI_SUBCLASS_SPECIFIC 0x00000010) = (0x03000000 0x000F0000) (0x1000 0x00000010)	0x030F1010
Instance	0		0x00

IPMI RAW COMMAND 0x2C 0x02 0xAE 0x01 0x00 0x00 0x00 0x10 0x10 0x0F 0x03 0x00

Example 3 – PCI bus resource allocation

UEFI Definition	EFI_STATUS_CODE_TYPE	EFI_PROGRESS_CODE	0x00000001
	EFI_STATUS_CODE_VALUE	EFI_IO_BUS_PCI EFI_IOB_PCI_RES_ALLOC = (EFI_IO_BUS 0x00010000) (EFI_SUBCLASS_SPECIFIC 0x00000001) = (0x02000000 0x00010000) (0x1000 0x00000001)	0x02011001
Instance		0	0x00

IPMI RAW COMMAND 0x2C 0x02 0xAE 0x01 0x00 0x00 0x00 0x01 0x10 0x01 0x02 0x00

Example 4 – Uncorrectable memory error on DIMM 2

UEFI Definition	EFI_STATUS_CODE_TYPE	EFI_ERROR_CODE	0x90000002
		ERROR_UNRECOVERED = 0x90	
	EFI_STATUS_CODE_VALUE	EFI_COMPUTING_UNIT_MEMORY EFI_CU_MEMORY_EC_UNCORRECTABLE = (EFI_COMPUTING_UNIT 0x00050000) EFI_SUBCLASS_SPECIFIC 0x00000003) = (0x00000000 0x00050000) (0x1000 0x00000003)	0x00051003
	Instance	2	0x02

IPMI RAW COMMAND 0x2C 0x02 0xAE 0x02 0x00 0x00 0x00 0x03 0x10 0x05 0x00 0x02

Example 5 – OEM specific I2C bus error on bus 4

UEFI Definition	EFI_STATUS_CODE_TYPE	EFI_ERROR_CODE	0x90000002
		ERROR_UNRECOVERED = 0x90	
	EFI_STATUS_CODE_VALUE	EFI_IO_BUS_I2C EFI_IO_PLATFORM_SPECIFIC_ERROR2 = EFI_IO_BUS 0x000C0000 (EFI_OEM_SPECIFIC 0x00000012) = (0x02000000 0x000C0000) (0x8000 0x00000012)	0x020C8012
	Instance	4	0x04

IPMI RAW COMMAND 0x2C 0x02 0xAE 0x02 0x00 0x00 0x00 0x12 0x80 0x0C 0x02 0x04

F.2.2 Example boot progress codes (Redfish)

DMTF Redfish Schema Supplement [7] [33], version 2020.3 and newer, introduced a method to read the last Boot Progress Code using the `ComputerSystem.BootProgress` Redfish object. Using this feature, the user can read the last Boot Progress Code that was reported by system firmware to the BMC. The DMTF schema defines a handful of standard boot progress codes and a method for reporting implementation-specific OEM defined codes.

SBMR recommends that Level M2.1-based server systems report the Boot Progress Codes through Redfish out-of-band and in-band interfaces. When possible, implementations should use the DMTF-defined standard codes. If the Boot Progress Code does not map to one of the DMTF defined codes, SBMR recommends reporting the codes as defined in Section F.2. Achieve this by setting the Redfish `BootProgress.LastState` property to `OEM` and setting the `BootProgress.OEMLastState` property to the 9-byte hex values defined in Section F.2.

Here is an example of the Redfish JSON mockup for Boot Progress property and how it maps to the UEFI and IPMI definitions:

Example 1 - Host processor power-on initialization

(Refer to IPMI Example 1 in Section F.2.1)

UEFI PI Status Code Definition:

EFI_COMPUTING_UNIT_HOST_PROCESSOR | EFI_CU_HP_PC_POWER_ON_INIT, Instance = 0

IPMI command to send the progress code to the BMC:

0x2C 0x02 0xAE 0x01 0x00 0x00 0x00 0x00 0x10 0x01 0x00 0x00

Redfish JSON mockup when reading the Progress Code from the Redfish interface:

```
{
  "BootProgress": {
    "LastState": "OEM",
    "OemLastState" : "0x010000000010010000",
    "LastStateTime": "2020-03-13T04:14:13+06:00",
  },
}
```

F.3 Common boot progress codes

The following table describes some common combinations of Boot Progress Codes and Boot Error Codes that can be used on SBMR Level M2.1-based server systems servers. For the raw values of these definitions, refer to PI Specification [35] and to Section F.2

Name	Progress Code
Driver eXecution Environment (DXE) Core started	EFI_SOFTWARE_DXE_CORE EFI_SW_DXE_CORE_PC_ENTRY_POINT
DXE Variable Block NVRAM init	EFI_SOFTWARE_EFI_BOOT_SERVICE BS_PC_NVRAM_INIT
DXE CPU Init Begin	EFI_COMPUTING_UNIT_HOST_PROCESSOR EFI_CU_PC_INIT_BEGIN
Powering on and Configuring CPU	EFI_COMPUTING_UNIT_HOST_PROCESSOR EFI_CU_HP_PC_POWER_ON_INIT
DXE CPU Init End	EFI_COMPUTING_UNIT_HOST_PROCESSOR EFI_CU_PC_INIT_END
DXE SoC Devices Init	EFI_COMPUTING_UNIT_CHIPSET EFI_CHIPSET_PC_DXE_SB_DEVICES_INIT
DXE handoff to UEFI Boot Device Selection (BDS) phase	EFI_SOFTWARE_DXE_CORE EFI_SW_DXE_CORE_PC_HANDOFF_TO_NEXT
BDS Connect UEFI Drivers	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_DXE_BS_PC_BEGIN_CONNECTING_DRIVERS
PCI Bus Init	EFI_IO_BUS_PCI EFI_IOB_PC_INIT
PCI Bus Enumeration	EFI_IO_BUS_PCI EFI_IOB_PCI_BUS_ENUM
PCI Bus Request Resources	EFI_IO_BUS_PCI EFI_IOB_PC_ENABLE
PCI Bus Assigned Resources	EFI_IO_BUS_PCI EFI_IOB_PC_ENABLE
Console Out Devices Connected	EFI_PERIPHERAL_LOCAL_CONSOLE EFI_P_PC_INIT
Input Devices connected	EFI_PERIPHERAL_KEYBOARD EFI_P_PC_INIT
USB Init	EFI_IO_BUS_USB EFI_IOB_PC_INIT
USB HotPlug	EFI_IO_BUS_USB EFI_IOB_PC_HOTPLUG
USB Device Detect	EFI_IO_BUS_USB EFI_IOB_PC_ENABLE
Serial ATA Init	EFI_IO_BUS_ATA_ATAPI EFI_IOB_PC_INIT
Serial ATA Detect	EFI_IO_BUS_ATA_ATAPI EFI_IOB_PC_DETECT
SCSI Init	EFI_IO_BUS_SCSI EFI_IOB_PC_INIT
SCSI Detect	EFI_IO_BUS_SCSI EFI_IOB_PC_DETECT
Fixed Media Init	EFI_PERIPHERAL_FIXED_MEDIA EFI_P_PC_INIT
Fixed Media Detect	EFI_PERIPHERAL_FIXED_MEDIA EFI_P_PC_PRESENCE_DETECT
Removable Devices Init	EFI_PERIPHERAL_REMOVABLE_MEDIA EFI_P_PC_INIT
Removable Devices Detect	EFI_PERIPHERAL_REMOVABLE_MEDIA EFI_P_PC_PRESENCE_DETECT
SMBUS Init	EFI_IO_BUS_SMBUS EFI_IOB_PC_INIT
I2C Init	EFI_IO_BUS_I2C EFI_IOB_PC_INIT

Name	Progress Code
Setup Verifying Password	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_DXE_BS_PC_VERIFYING_PASSWORD
Setup Start	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_PC_USER_SETUP
Setup Input Wait	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_PC_INPUT_WAIT
UEFI Ready to Boot Event	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_DXE_BS_PC_READY_TO_BOOT_EVENT
UEFI Exit Boot Services	EFI_SOFTWARE_EFI_BOOT_SERVICE EFI_SW_BS_PC_EXIT_BOOT_SERVICES
UEFI Exit Boot Services Event	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_DXE_BS_PC_EXIT_BOOT_SERVICES_EVENT
Set Virtual Address Map Begin	EFI_SOFTWARE_EFI_RUNTIME_SERVICE EFI_SW_RS_PC_SET_VIRTUAL_ADDRESS_MAP
Set Virtual Address Map End	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_DXE_BS_PC_VIRTUAL_ADDRESS_CHANGE_EVENT
Reset System	EFI_SOFTWARE_EFI_RUNTIME_SERVICE EFI_SW_RS_PC_RESET_SYSTEM

Error Codes

Name	PI Status
DXE Arch protocol is not available	EFI_SOFTWARE_DXE_CORE EFI_SW_DXE_CORE_EC_NO_ARCH
PCI Out Of Resources	EFI_IO_BUS_PCI EFI_IOB_EC_RESOURCE_CONFLICT
No Console Out	EFI_PERIPHERAL_LOCAL_CONSOLE EFI_P_EC_NOT_DETECTED
No Console In	EFI_PERIPHERAL_KEYBOARD EFI_P_EC_NOT_DETECTED
Invalid Password	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_DXE_BS_EC_INVALID_PASSWORD
Boot Option Failed	EFI_SOFTWARE_DXE_BS_DRIVER EFI_SW_DXE_BS_EC_BOOT_OPTION_FAILED
HDD SMART Error	EFI_IO_BUS_ATA_ATAPI EFI_IOB_ATA_BUS_SMART_OVERTHRESHOLD
Flash not available	EFI_COMPUTING_UNIT_MEMORY EFI_CU_MEMORY_EC_UPDATE_FAIL