Arm® Server Base Manageability Requirements 2.0

Platform Design Document

Non-confidential

Document number: DEN0069D

Release information

Date	Version	Changes
27 Apr 2022	Issue D (2.0)	• Finalize Level M3 (351). • Finalize Level M4 (352). • Add SPDM and MCTP security requirements (347). • Add optional SoC-BMC UART for DBG2 (366). • Add BMC initiated firmware updates guidance (304). • Update M3/M4 OOB requirements (343). • Update M3/M4 in-band requirements (344). • Update M3/M4 side-band requirements (345). • Update M3/M4 BMC-IO requirements (346). • Add CXL management requirements to M4 (341). • Update M3/M4 BMC-Platform Elements requirements (398). • Update JTAG connectivity requirements and security considerations (437). • Update PLDM platform monitoring (353). • Update RAS PLDM logging flows (349). • Update use cases and background, and remove MCTP Host Interface (399). • PCIe x1 security considerations (481). • Update OCP Redfish Profile reference (471). • Clarify RAS CPER format (414). • Fix Send Platform Error Record IPMI command Response data (433). • Clarify IPMI usage for in-band RAS event logging (439). • Reference DC-SCM specification (350). • Update DMTF specification references (402). • Add PMCI Architecture whitepaper reference (403). • Remove SBSG reference (400). • Inclusive language considerations (401).
11 Feb 2021	Issue C (1.1)	• 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 (1.0)	• License LES-PRE-21585
30 Jan 2020	Issue A (1.0)	• 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.

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.

Contents

Release information 2
Arm Non-Confidential Document Licence (“Licence”) 3

About this document 9
Terms and abbreviations 9
References 10
Cross References 12
Rules-based writing 12
Identifiers 12
Examples 13
Feedback 13

1 Scope and Background 14
1.1 Scope 14
1.2 Background 15
1.2.1 Host SoC in-band interface 15
1.2.2 SoC side-band interface 16
1.2.3 PCIe connection between the Arm SoC and the BMC 16
1.2.4 USB connection between the Arm SoC and the BMC 17
1.2.5 JTAG connection between the Arm SoC and the BMC 17
1.2.6 Additional connectivity between the Arm SoC and the BMC 17
1.2.7 Multi-socket platform 17
1.3 Arm SoC-BMC interface terminology 18

2 Compliance Levels and Requirements 20
2.1 Level M0 23
2.2 Level M1 24
2.2.1 SoC-BMC interface 25
2.2.2 BMC-platform elements interface 26
2.2.3 BMC management services (out-of-band) interface 26
2.3 Level M2 27
2.3.1 SoC-BMC interfaces 28
2.3.2 BMC-platform elements interface 28
2.3.3 BMC-IO device interface 28
2.3.4 BMC management services (out-of-band) interface 29
2.4 Level M2.1 30
2.4.1 SoC-BMC interfaces 31
2.4.2 BMC-platform elements interface 31
2.4.3 BMC-IO device interface 32
2.4.4 BMC management services (out-of-band) interface 32
2.5 Level M3 33
2.5.1 SoC-BMC interface 34
2.5.2 BMC-platform elements interface 35
2.5.3 BMC-IO device interface 35
2.5.4 BMC management services (out-of-band) interface 36
2.5.5 SPDM over MCTP for BMC and side-band devices 36
2.6 Level M4 38
2.6.1 SoC-BMC interface 39
2.6.2 BMC-IO device interface 39
2.6.3 BMC-platform elements interface 40
2.6.4 BMC management services (out-of-band) interface 40
2.6.5 SPDM over MCTP for BMC and side-band devices 40
2.7 SBMR checklist 41

2.7.1 SBMR Level M1 checklist 41
2.7.2 SBMR Level M2 checklist 41
2.7.3 SBMR Level M2.1 checklist 42
2.7.4 SBMR Level M3 checklist 42
2.7.5 SBMR Level M4 checklist 43

A OpenBMC 44

B IPMI 45
B.1 Standard IPMI commands 45
B.1.1 Remote power control 45
B.1.2 Boot device selection 45
B.1.3 BMC to Host mapping 45
B.1.4 BMC user manipulation 46
B.1.5 Redfish host interface credentials bootstrapping 46
B.1.6 IPMI support verification 46
B.2 Arm standard IPMI commands 46
B.2.1 General IPMI commands format 46
B.2.2 List of Arm standard IPMI commands 47
B.3 IPMI specification clarifications and corrections 47
B.4 SSIF single and multi-part transactions 48

C RAS 50
C.1 Level M0 50
C.2 Level M1 50
C.2.1 SMBus System Interface (SSIF) in-band interface 51
C.2.2 RAS IPMI message format 52
C.2.3 SoC side-band interface 52
C.2.4 Out-of-band interface 52
C.3 Level M2 and Level M2.1 52
C.3.1 Redfish and IPMI host (in-band) interfaces 53
C.3.2 RAS Redfish message format 53
C.3.3 SoC side-band interface 54
C.3.4 Out-of-band interface 54
C.4 Level M3 and M4 54
C.4.1 Redfish host (in-band) interface 55
C.4.2 MCTP and PLDM (SoC side-band) interface 55
C.4.3 RAS PLDM message format 57
C.4.4 Out-of-band interface 63

D Platform Monitoring and Control 65
D.1 Introduction 65
D.2 IPMI commands to monitor and control managed entities 65
D.3 Redfish schema to monitor and control managed entities 66
D.4 PLDM commands/APIs to monitor and control managed entities 66
D.4.1 Examples of PLDM sensors exposed by SatMC 70

E Reference Implementation of Remote Debug Using OpenOCD 71
E.1 Introduction 71
E.2 Levels M1, M2, M2.1, M3, M4 71

F Boot Progress Codes 73
F.1 IPMI commands for boot progress codes 73
F.1.1 Send boot progress code (NetFn 2Ch, Command 02h) 73
F.1.2 Get boot progress code (NetFn 2Ch, Command 03h) 73
F.2 Boot progress code format 74

F.2.1 Example progress codes (IPMI) 75
F.2.2 Example boot progress codes (Redfish) 77
F.3 Common boot progress codes 77

G Trusted Communication Between MC and Server System Devices 80
G.1 MC and server system device attestation 80
G.2 MC and server system device mutual attestation 80
G.3 MC and server system device measurement 80
G.4 Data encryption between MC and server system device 81

H Firmware Update 82
H.1 Host-based firmware update 82
H.2 BMC-based firmware update 82
H.3 Firmware inventory 83

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.
Completer	An agent in a computing system that responds to and completes a memory transaction that was initiated by a Requester.
CXL FM	CXL Fabric Manager
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.
Management Controller (MC)	A microcontroller or processor with a platform or SoC specific device management functionality. Management Controller may include multiple physical interfaces and implement various types of protocols for communication with managed devices, application processors or other MCs. See BMC and SatMC for MC examples.
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.

Term	Meaning
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.
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.
Requestor	An agent in a computing system that is capable of initiating memory transactions.
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.
SBSA	Server Base System Architecture.
SiP	Silicon Partner. In this document, the silicon manufacturer.
SMBIOS	System Management BIOS
SPDM	Security Protocol and Data Model. A data model that defines messages, data objects, and sequences for performing message exchanges between devices over a variety of transport and physical media. The description of message exchanges includes authentication of hardware identities and measurement for firmware identities. The SPDM enables efficient access to low-level security capabilities and operations. The SPDM can be used with other mechanisms, including non-PMCI- and DMTF-defined mechanisms.
UEFI	Unified Extensible Firmware Interface.

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 (ACPI) Specification*. UEFI.org.

[6] *Unified Extensible Firmware Interface (UEFI) Specification*. UEFI.org.

[7] *DSP8010 Redfish Schema*. DMTF.

[8] *Intelligent Platform Management Interface (IPMI) 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] *DSP2015 PMCI Architecture White Paper*. DMTF.

[12] *DSP0134 System Management BIOS (SMBIOS) Reference Specification*. DMTF.

[13] *DSP0256 MCTP Host Interface Specification*. DMTF.

[14] *DSP0238 MCTP PCIe VDM Transport Binding Specification*. DMTF.

[15] *DEN 0101 Authenticated Debug Access Control Specification (ADAC)*. Arm Ltd.

[16] *DEN 0094 Arm Base System Architecture (BSA)*. Arm Ltd.

[17] *OCP Server Designs and Specifications*. Open Compute Project.

[18] *OCP Datacenter Secure Control Module*. Open Compute Project.

[19] *DEN 0044 Arm Base Boot Requirements (BBR)*. Arm Ltd.

[20] *Arm IHI 0031 Arm Debug Interface Architecture Specification, ADIv5*. Arm Ltd.

[21] *Arm IHI 0074 Arm Debug Interface Architecture Specification, ADIv6*. Arm Ltd.

[22] *DSP0270 Redfish Host Interface Specification*. DMTF.

[23] *DSP0222 NC-SI Specification*. DMTF.

[24] *DSP0272 Redfish Interoperability Profile Specification*. DMTF.

[25] *DSP8013 Redfish Interoperability Profiles Bundles*. DMTF.

[26] *DSP2046 Redfish Resource and Schema Guide*. DMTF.

[27] *DSP0245 PLDM IDs and Codes Specification*. DMTF.

[28] *DSP0248 PLDM for Platform Monitoring and Control Specification*. DMTF.

[29] *DSP0249 PLDM State Set Specification*. DMTF.

[30] *DSP0239 MCTP IDs and Codes*. DMTF.

[31] *DSP0241 PLDM Over MCTP Binding Specification*. DMTF.

[32] *DSP0274 Security Protocol and Data Model (SPDM) Specification*. DMTF.

[33] *DSP0275 Security Protocol and Data Model (SPDM) over MCTP Binding Specification*. DMTF.

[34] *DSP0277 Secured Messages using SPDM Specification*. DMTF.

[35] *DSP0276 Secured Messages using SPDM over MCTP Binding Specification*. DMTF.

[36] *DSP0237 MCTP SMBus/I2C Transport Binding Specification*. DMTF.

[37] *DSP0233 MCTP I3C Transport Binding Specification*. DMTF.

[38] *DSP0267 PLDM for Firmware Update Specification*. DMTF.

[39] *DSP0218 PLDM for Redfish Device Enablement (RDE) Specification*. DMTF.

[40] *NVM Express Management Interface*. NVM Express.

[41] *DSP0235 NVMe Management Messages over MCTP Binding Specification*. DMTF.

[42] *OCP Usage Guide for Server Profile*. Open Compute Project.

[43] *CXL 2.0 specification*. computeexpresslink.org.

[44] *CXL Type 3 Management Using MCTP CCI ECN*. computeexpresslink.org.

[45] *DSP0234 CXL Fabric Manager API over MCTP Binding Specification*. DMTF.

[46] *DSP0281 CXL Type 3 Device Component Command Interface over MCTP Binding Specification*. DMTF.

[47] *DSP0268 Redfish Schema Supplement*. DMTF.

[48] *OpenOCD Project*. OpenOCD Project.

[49] *Platform Initialization (PI) Specification*. UEFI.org.

[50] *DEN 0118 Platform Security Firmware Update for the A-profile Arm Architecture*. Arm Ltd.

[51] *DSP2062 Redfish Firmware Update White Paper*. DMTF.

[52] *DSP0261 NC-SI over MCTP Binding Specification*. DMTF.

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

Rules-based writing

This specification consists of a set of individual rules. Each rule is clearly identified by the letter R.

Rules must not be read in isolation, and where more than one rule relating to a particular feature exists, individual rules are grouped into sections and subsections to provide the proper context. Where appropriate, these sections contain a short introduction to aid the reader. An implementation which is compliant with the architecture must conform to all of the rules in this specification.

Some architecture rules are accompanied by rationale statements which explain why the architecture was specified as it was. Rationale statements are identified by the letter X.

Some sections contain additional information and guidance that do not constitute rules. This information and guidance is provided purely as an aid to understanding the architecture. Information statements are clearly identified by the letter I.

Implementation notes are identified by the letter U.

Software usage descriptions are identified by the letter S.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Rules, rationale statements, information statements, implementation notes and software usage statements are collectively referred to as *content items*.

Identifiers

Each content item may have an associated identifier which is unique within the context of this specification.

When the document is prior to beta status:

- Content items are assigned numerical identifiers, in ascending order through the document (*0001*, *0002*, . . .).
- Identifiers are volatile: the identifier for a given content item may change between versions of the document.

After the document reaches beta status:

- Content items are assigned random alphabetical identifiers (*HJQS*, *PZWL*, . . .).
- Identifiers are preserved: a given content item has the same identifier across versions of the document.

Examples

Below are examples showing the appearance of each type of content item.

R This is a rule statement.

R_{X001} This is a rule statement.

I This is an information statement.

X This is a rationale statement.

U This is an implementation note.

S This is a software usage description.

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 (DEN0069D 2.0).
- 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 is 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.

PLDM and MCTP are industry standards targeting “inside the box” communication. They are defined by the DMTF Platform Management Component Intercommunication (PMCI) Working Group. For an overview of the PMCI management stack, refer to the DSP2015 - PMCI Architecture White Paper [11]. Figure 3 in that white paper shows a detailed diagram of the relationship of each specification in the PMCI Stack, including MCTP and PLDM.

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

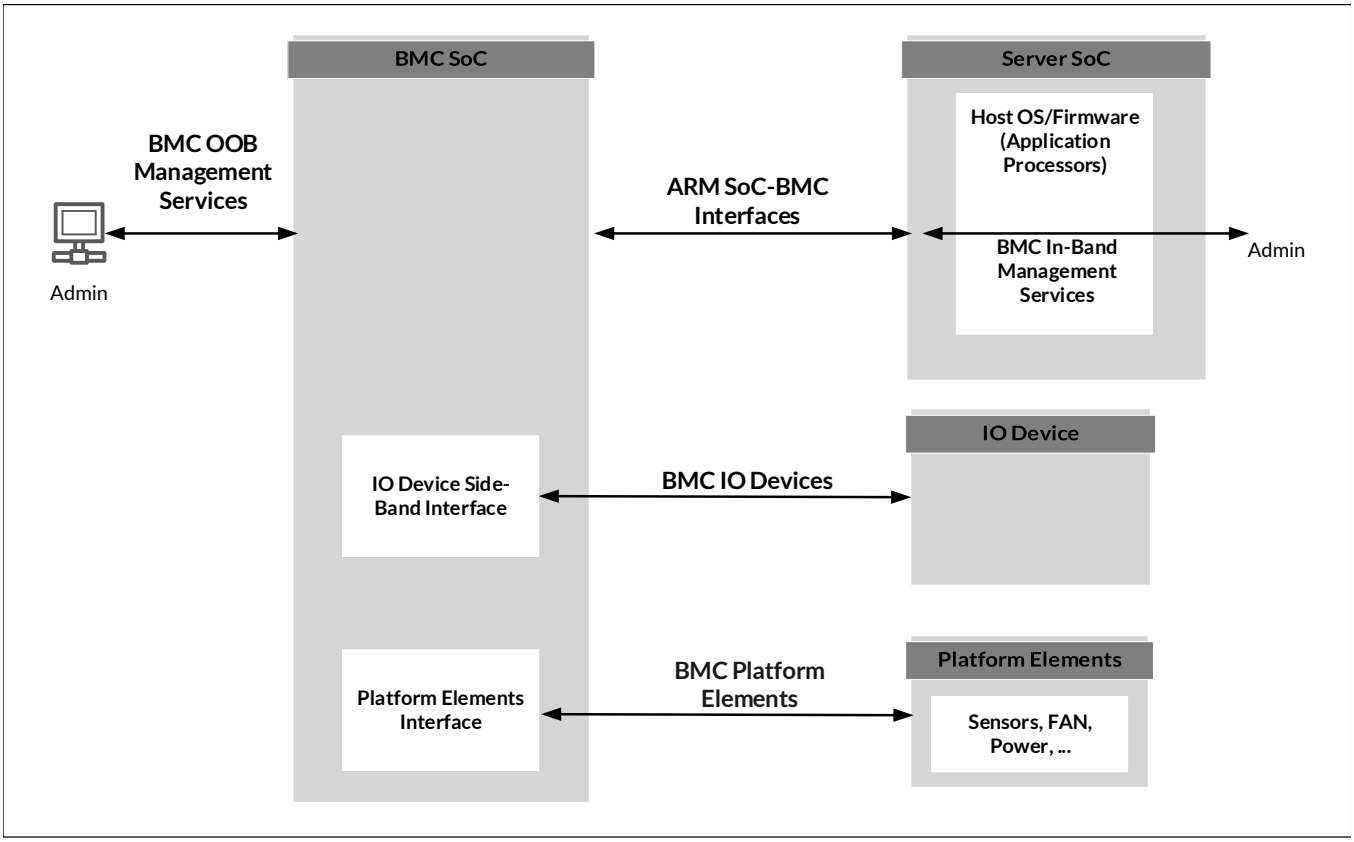

Figure 1: Server Management Interfaces

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 server management interfaces, as described in the Table 3 summary below.

This document may also provide some requirements, recommendations, and guidance 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 [12], ACPI [5] tables (e.g. SPMI), and/or PCIe configuration space. Arm server systems typically use the IPMI SSIF Host Interface[8], with newer Arm server systems adding Redfish Host Interface [2]. Future Arm Server systems may transition to MCTP host interface [13] as an IPMI SSIF replacement.

Typical use cases of this interface include:

- UEFI - BMC communication, using IPMI OEM (and possibly standard) commands.
  - Reporting SMBIOS [12] table
  - Reporting boot progress codes

  - Error reporting (in some cases)
  - General event logging
  - General UEFI - BMC data exchange

Note

Sending large amount of data (such as SMBIOS table) over the slow IPMI SSIF I2C bus during boot may impact the boot time. Until a suitable higher bandwidth standard interface is defined, implementations may choose alternative non-standard interfaces for these use-cases, including using a proprietary interface, such as a PCIe-based mailbox or shared memory. Redfish Host Interface may also be used. Even though that interface is intended for OS/Hypervisor and User software communication with the BMC, it can be used for some UEFI to BMC communication using non-standard OEM Redfish schema extensions.

Note

These use cases may transition in the future to using a standard and higher bandwidth interface, such as MCTP Host Interface.

- OS/Hypervisor software – BMC communication
  - Redfish Authentication, using IPMI SSIF, and possibly MCTP or other interfaces on future Arm server systems.
- User software – BMC communication
  - User or Admin access to BMC management services, using IPMI SSIF and/or Redfish Host Interface, 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 and power
- RAS error reporting
- Early stages of boot event logging

Note

It is also possible for this interface to be used for some of the use cases of UEFI – BMC communication, as an alternative path to the Host SoC in-band interface.

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 non-standard memory mailbox communication between the BMC and the SoC host software.

Note

Security must be considered when using this interface to ensure isolation of host and BMC security domains. For example, untrusted users that have access to the host software must not be able to access privileged BMC resources, such as firmware storage.

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, USB Disk)
- 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 necessarily 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. This may include for example: a hardware fuse, hardware jumper, protected firmware setting, or using an authenticated debug mechanism, such as the Arm Authenticated Debug Access Control (ADAC) [15].

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. The terms relevant to the areas covered are defined in this section.

Table 3: Arm SoC-BMC Interface Terminology

Name	Requester	Completer	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 other proprietary interface. • This interface is invasive to the main processor complex (i.e. processing cycles are required).	Yes
SoC Side-Band Interface	BMC	SoC / SatMC	• This interface may leverage a proprietary protocol or a more standard transport protocol, such as MCTP/PLDM. • This is a multi-master bi-directional communication interface. • 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 SoC.	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. • Some pins may be bi-directional (e.g. PROCHOT)	Partially Covered

Name	Requester	Completer	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. • 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 BSA [16] 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, PCIe cards, and NVMe disks. • These interfaces are only partially in scope of the SBMR compliance requirements. Some requirements, recommendations and guidance may be provided based on external specifications and standards, such as MCTP/PLDM.	Partially Covered
Misc Platform elements (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 • 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.	Partially Covered

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, M3, and M4. 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.

Table 4 below shows the summary of SBMR Compliance levels.

This table is indicative only. The rules in each level describe the specific features that are required to be compliant to that level. For a checklist of each level's minimum rules, refer to Section 2.7.

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 is used, NC-SI is required.	IMPLEMENTATION DEFINED

Level	Out-of-band Interface	SoC Side-band Interface	Host/SoC In-band Interface	BMC IO Device Interface	BMC Platform Element Interface
M3	**Required**: Redfish.	**Required**: MCTP/PLDM over I2C/SMBus or a higher bandwidth interface.	**Required**: IPMI SSIF and Redfish Host Interface.	**Conditional Requirement**: If shared physical NIC is used, NC-SI over RBT or MCTP (over I2C/SMBus or a higher bandwidth interface). **Recommended**: MCTP/PLDM for PCIe devices (Network and Storage), and NVMe-MI over MCTP (for NVMe disks), using I2C/SMBus or a higher bandwidth interface. **IMPLEMENTATION DEFINED** Other IO Devices	IMPLEMENTATION DEFINED Refer to [17], [18] and [8] for guidance.

Level	Out-of-band Interface	SoC Side-band Interface	Host/SoC In-band Interface	BMC IO Device Interface	BMC Platform Element Interface
M4	**Required**: Redfish.	**Required**: MCTP/PLDM over I3C.	**Required**: IPMI SSIF and Redfish Host Interface.	**Conditional Requirement**: If shared physical NIC is used, NC-SI (over I3C or PCIe VDM). **Conditional Requirement**: MCTP/PLDM for PCIe devices (Network and Storage), and NVMe-MI over MCTP (for NVMe disks), using I3C or PCIe VDM, with I2C as fallback. **Recommended**: CXL FM and CCI over MCTP for CXL devices, using I2C or PCIe VDM, with I2C as fallback. Other IO Devices **IMPLEMENTATION DEFINED**.	IMPLEMENTATION DEFINED Refer to [17], [18] and [8] for guidance. **Recommended**: PLDM/MCTP

2.1 Level M0

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

I 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

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

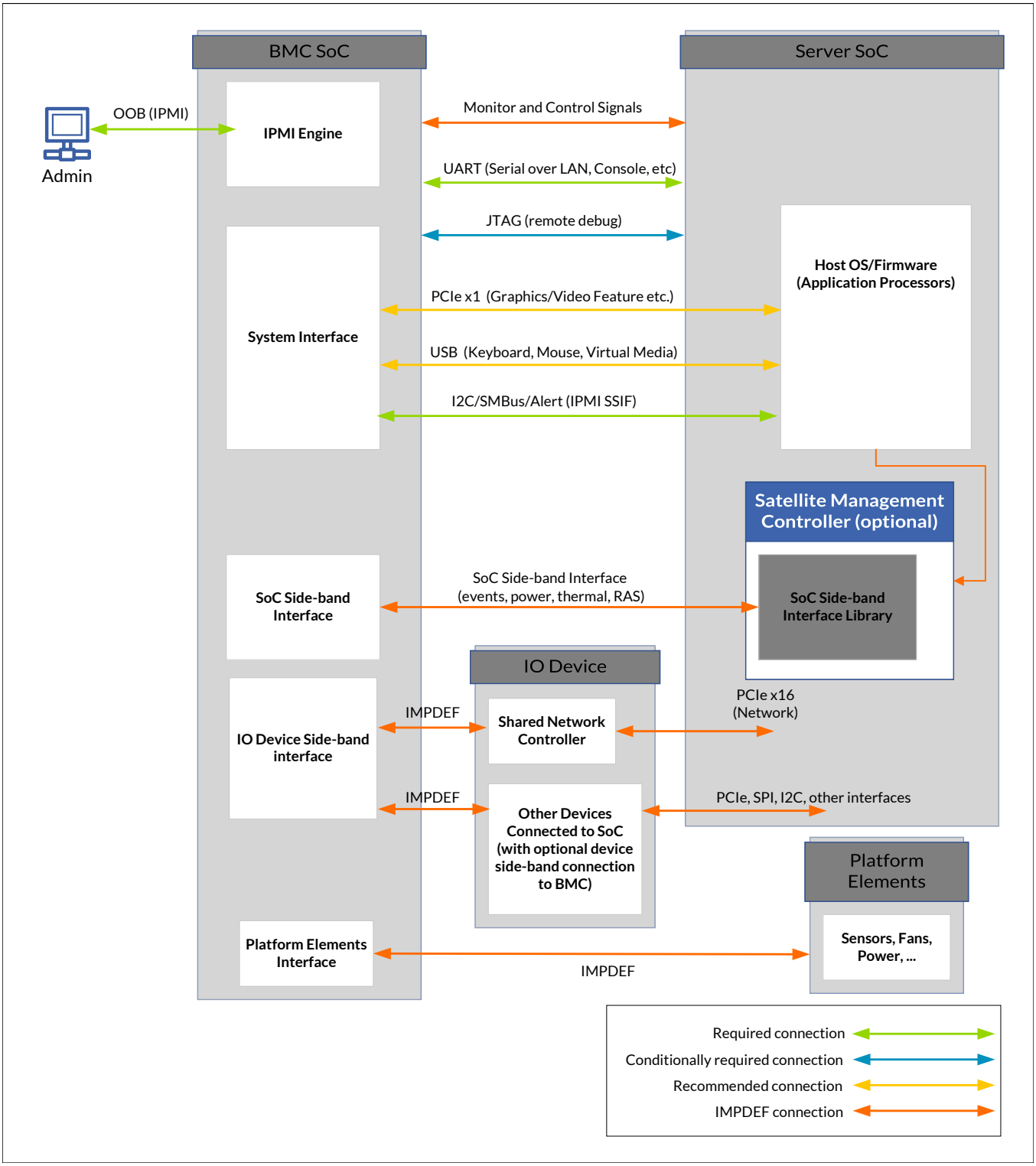

Figure 2: Server Management Interfaces (Level M1)

2.2.1 SoC-BMC interface

R Most SoC-BMC interfaces for the Level M1-based server systems are IMPLEMENTATION DEFINED, with the exceptions of the requirements and recommendations described in the following subsections.

2.2.1.1 Host SoC in-band interface

$R_{M1\_IB\_1}$ 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 SMBus System Interface (SSIF) as the interface for IPMI in-band communication. The Arm SoC must have an SSIF connection to the BMC for IPMI communication as described by the IPMI specification. At a minimum, this must be an I2C connection used for sending IPMI commands to the BMC.

I It is recommended that an ALERT pin is also supported to enable BMC notification to the host.

I 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.

I Standard RAS error logging support for level M1 servers is described in Section C.2.

2.2.1.2 Console UART

$R_{M1\_UART\_1}$ The Arm SoC must have at least one BSA [16] 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.

$R_{M1\_UART\_2}$ Per the BSA [16] and BBR [19], the console UART must be a BSA [16] compliant UART that is 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.

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

2.2.1.3 PCIe

I 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.

2.2.1.4 USB

I 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.

2.2.1.5 JTAG

X 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.

$R_{M1\_JTAG\_1}$ 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.

I 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.

$R_{M1\_JTAG\_2}$ 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 [20] or ADIv6 [21].

- 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.

U For a reference implementation and more details, refer to Section E.

2.2.2 BMC-platform elements interface

I 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

$R_{M1\_OOB\_1}$ Support for IPMI is a requirement for M1-compliant server systems.

R Refer to Section B.1 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.

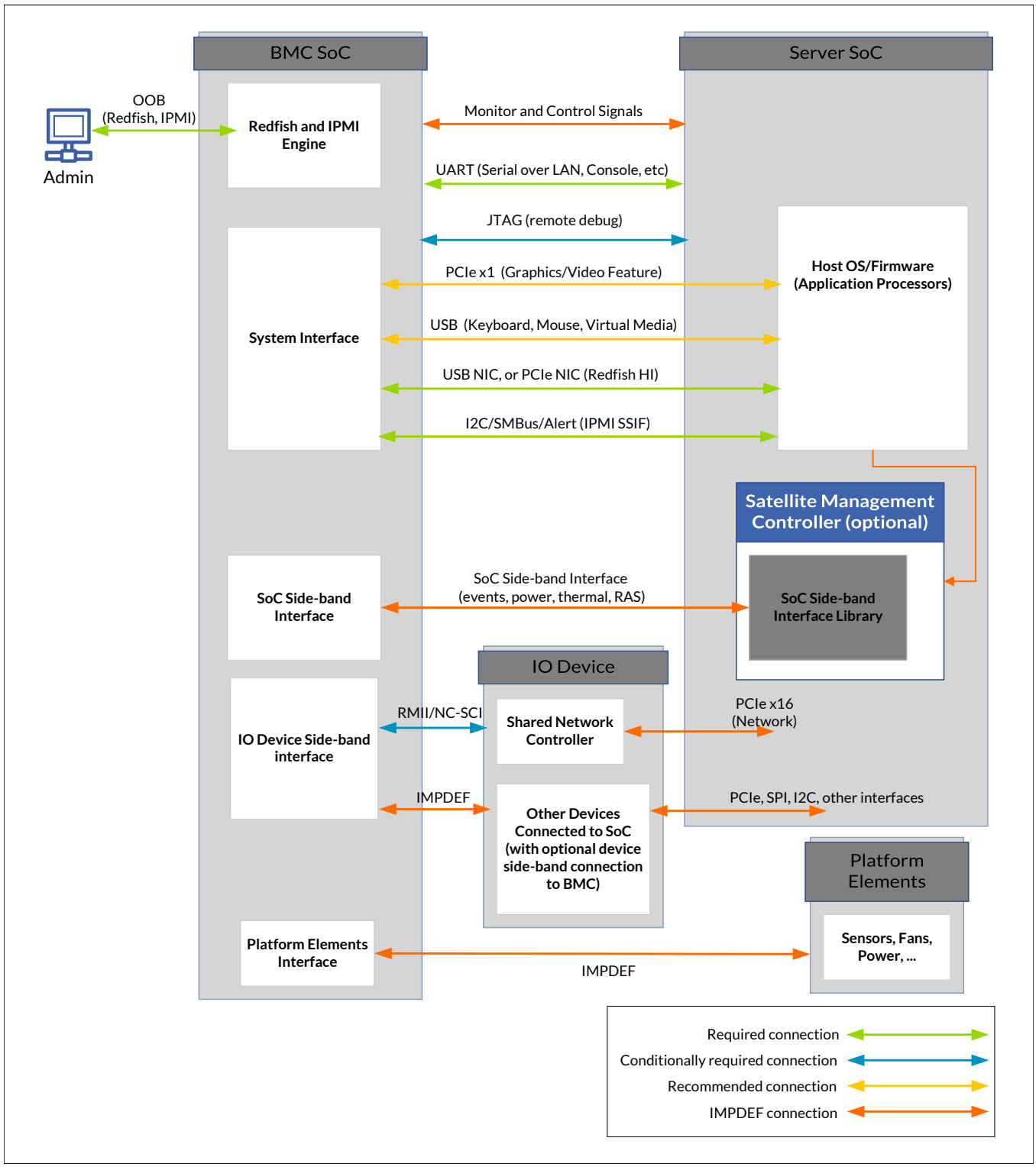

Figure 3: Server Management Interface (Level M2)

2.3.1 SoC-BMC interfaces

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

2.3.1.1 Host SoC in-band interface

$R_{M2\_IB\_1}$ The Host/SoC In-Band interface must be compliant to the Redfish Host Interface Specification [22]. 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.

I In addition to USB or PCIe network device, [22] defines an OEM proprietary method. This proprietary method is not recommended for M2-compliant systems.

$R_{M2\_IB\_2}$ In addition to the Redfish Host Interface, M2-compliance requires that a second Host-SoC in-band interface based on IPMI must exist.

2.3.1.2 JTAG

$R_{M2\_JTAG\_1}$ JTAG connection between the BMC and the SoC remains a conditional requirement in Level M2-based server systems if support for JTAG-based remote debug and crash dump functions is needed.

$R_{M2\_JTAG\_2}$ In addition, SBMR Level-M2 compliant SoC and BMC silicon parts are conditionally required to provide the JTAG debug capability if support for JTAG-based remote debug and crash dump functions is needed. This is to allow for systems to optionally implement the SoC-BMC JTAG connection using these parts.

2.3.2 BMC-platform elements interface

I The BMC-Platform Elements interface for the Level M2-based server systems is IMPLEMENTATION DEFINED. Typically, the SMBus/I2c medium is used.

2.3.3 BMC-IO device interface

$R_{M2\_IO\_1}$ When using a shared physical NIC interface between the BMC and the Arm SoC, then Network Controller Side-band Interface (NC-SI)[23] over reduced media independent interface (RMII) based transport is required for Level M2-based server systems.

X NC-SI[23] 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.

I 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

$R_{M2\_OOB\_1}$ Level M2-based server systems requires that the BMC management services interface supports the Redfish Interface [2] .

$R_{M2\_OOB\_2}$ IPMI support is also a requirement for M2-compliant server systems.

R Refer to Section B.1 for the minimal IPMI commands required.

$R_{M2\_OOB\_3}$ Level M2-based server systems further standardize the BMC management services interface by adopting the Redfish Interoperability Profiles Specification [24] and the individual profiles contained in the Redfish Interoperability Profiles Bundle [25]. 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.

X 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.

I Redfish Resource and Schema Guide [26] provides information on how to use the Redfish API, targeted at consumption of the API.

S 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.

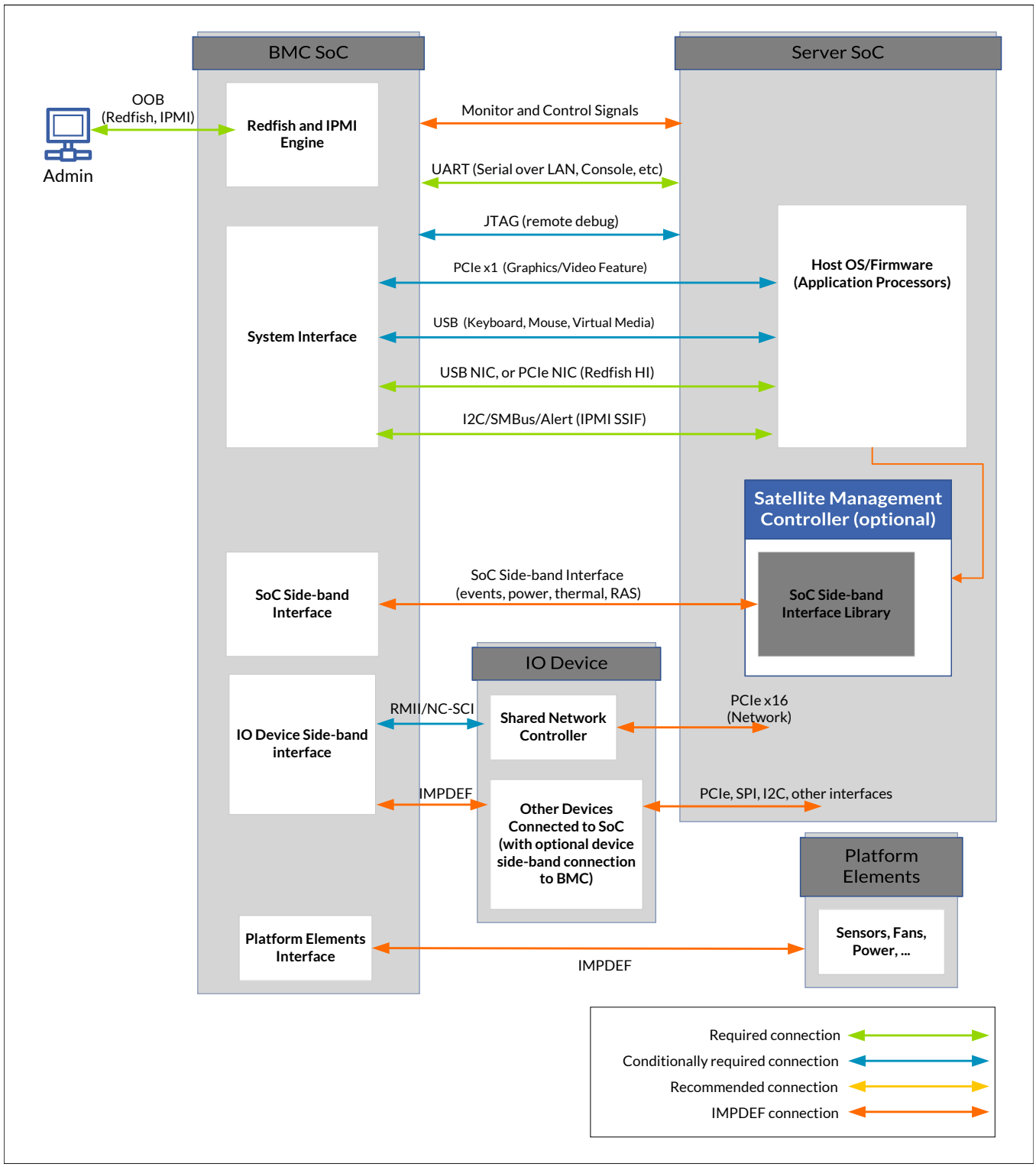

Figure 4: Server Management Interface (Level M2.1)

2.4.1 SoC-BMC interfaces

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

2.4.1.1 Host SoC in-band interface

$R_{M21\_IB\_1}$ The In-Band SSIF interface must follow the IPMI specification clarifications that are outlined in Section B.3 and Section B.4 of this specification.

$R_{M21\_IB\_2}$ 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.

I 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.

R_{M21_IPMI1} Level M2.1-based server systems must implement the following industry standard IPMI 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.

R_{M21_IPMI2} Level M2.1-based server systems must implement the following Arm-defined IPMI commands:

- Send Platform Error Record Section C.2.2. 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.

R If the platform supports reporting platform errors to the BMC using the in-band interface, then the additional rules in Section C.2 and Section C.3 must be implemented.

2.4.1.2 PCIe

$R_{M21\_PCI\_1}$ In levels M1 and M2, 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.

2.4.1.3 USB

$R_{M21\_USB\_1}$ In levels M1 and M2, 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.

2.4.2 BMC-platform elements interface

R 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

R 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

R 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

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

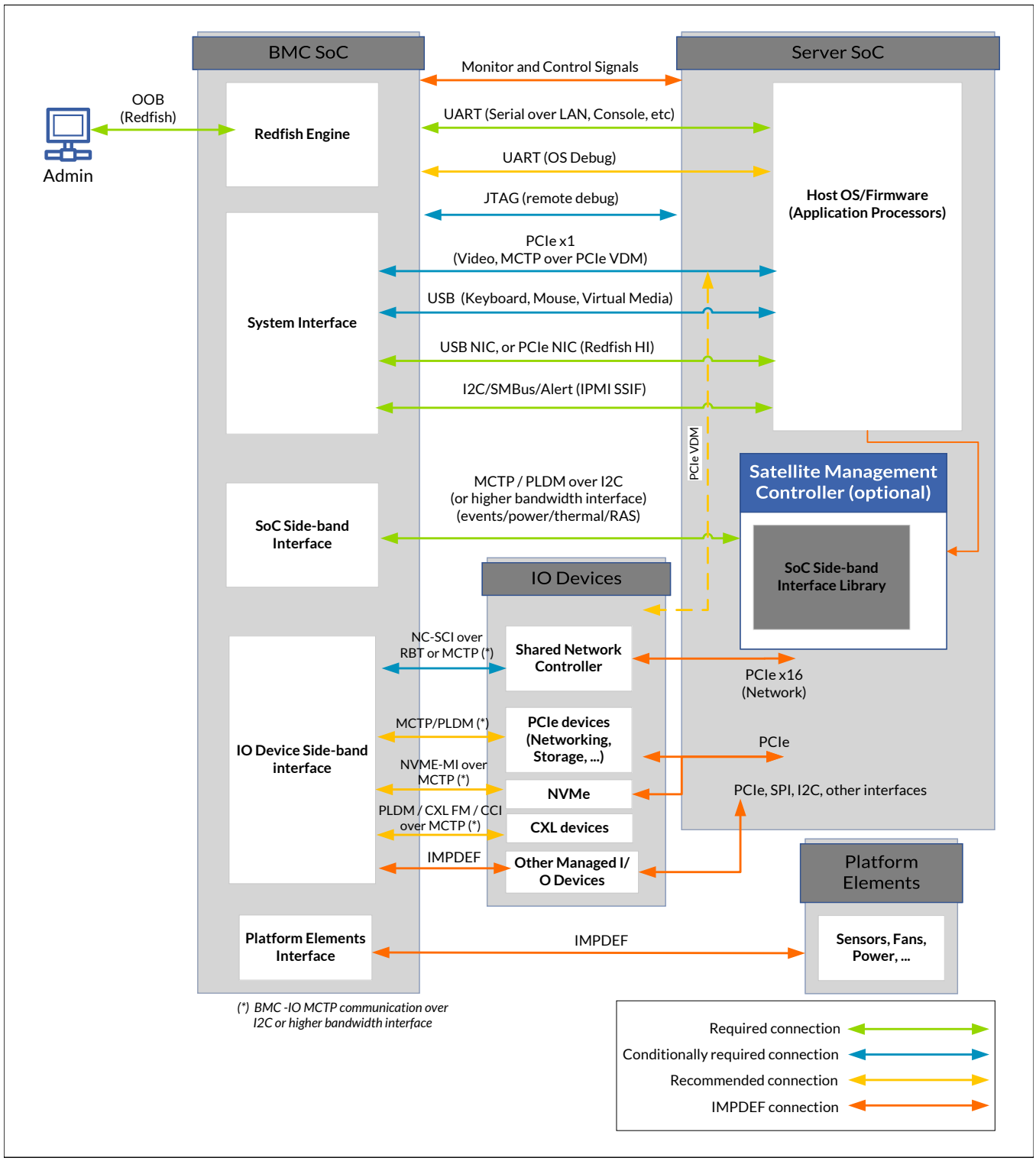

Figure 5: Server Management Interface (Level M3)

2.5.1 SoC-BMC interface

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

I For OCP compliant servers with modular design that use a BMC daughter card, SBMR recommends adoption of the Datacenter Secure Control Module 1.0 (DC-SCM)[18].

2.5.1.1 Host SoC in-band interface

R The requirements and recommendations for the Host/SoC In-Band interfaces for Level M3-based server systems are the same requirements and recommendations as for Level M2.1-based server systems.

2.5.1.2 Debug UART

I For Level M3 based systems, it is recommended that the Arm SoC has an additional BSA [16] compliant UART connection to the BMC for the purpose of remote OS debugging through the BMC.

I Per the BSA [16] and BBR [19], the debug UART must be a BSA [16] compliant UART that must be exposed to the host software using the Debug Port Table (DBG2) ACPI [5] Table. The default baud rate for interoperability with commercially available BMCs should be 115200 bits/second.

2.5.1.3 BMC-SoC Side-Band

$R_{M3\_SB\_1}$ 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).

$R_{M3\_SB\_2}$ PLDM [3] [27] [28] [29] is used for the purpose of supporting platform-level data models and platform functions.

X 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.

$R_{M3\_SB\_3}$ MCTP [4] [30] is used 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.

$R_{M3\_SB\_4}$ PLDM over MCTP binding [31] is used as the format of PLDM over MCTP messages.

$R_{M3\_SB\_5}$ SPDM [32] is used for the purpose of supporting security related capabilities of the devices.

X SPDM is designed to provide runtime authentication of a device by retrieving the certificate chains from it and verifying device authenticity by sending unique challenges. SPDM allows the requester to query the device’s firmware or configuration data measurements for device attestation purposes.

$R_{M3\_SB\_6}$ SPDM over MCTP binding [33] is used as the format of SPDM over MCTP messages.

$R_{M3\_SB\_7}$ Secure messages using SPDM specifications [34] is used for the purpose of supporting secure transfer of application data over PMCI transports using SPDM. Secure messages [34] also define the transport requirements for SPDM records, which form the basis of encryption and message authentication.

$R_{M3\_SB\_8}$ Secured Messages using SPDM over MCTP binding [35] is used as the format of SPDM secure messages over MCTP messages.

$R_{M3\_SB\_9}$ For Level M3 based server systems, the physical and data-link layer methods for MCTP communication are minimally defined by the MCTP over SMBus/I2C binding specification [36]. Implementations may choose a higher bandwidth physical data-link, such as MCTP over PCIe VDM [14] or MCTP over I3C [37]

Note

MIPI Alliance membership may be required to have full access and implementation rights to the I3C specifications.

U For Level M3 based server systems, SBMR recommends that PLDM is used for side-band interface BMC-SoC communication, as illustrated in Section D.4 and Section C.4.

2.5.1.4 JTAG

$R_{M3\_JTAG\_1}$ JTAG connection between the BMC and the SoC remains a conditional requirement in Level M3-based server systems.

$R_{M3\_JTAG\_2}$ If JTAG is implemented, the following requirements apply for systems used in production environments:

- The system must implement an IMPLEMENTATION DEFINED method to disable the JTAG connection between the BMC and the SoC.
- The system must implement an IMPLEMENTATION DEFINED method to disable JTAG remote access (from the BMC) to one or more subsystems.
- The system may implement an IMPLEMENTATION DEFINED method to re-enable JTAG connection between the BMC and the SoC.
- The system may implement an IMPLEMENTATION DEFINED method to re-enable JTAG (from the BMC) to one or more subsystems.

U Examples of such methods to disable/enable JTAG in production systems are described in section Section 1.2.5.

2.5.2 BMC-platform elements interface

I The BMC-Platform Elements interface for the Level M3-based server systems is IMPLEMENTATION DEFINED, with additional recommendations and guidance. Please refer to:

- Intelligent Platform Management Interface v2.0 (IPMI) specification [8].
- OCP server design and specifications [17].
- OCP Datacenter Secure Control Module (DC-SCM) [18].

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

2.5.3 BMC-IO device interface

$R_{M3\_IO\_1}$ If using shared physical NIC interface between BMC and SoC, then Network Controller Side-band Interface (NC-SI)[23] over reduced media independent interface (RMII) based transport or MCTP is required for Level M3 based server systems.

I Level M3 based server systems are recommended to standardize this interface based on the DMTF PMCI workgroup standards, which define specifications for primary intercommunication interfaces/data models between the Management Controller (BMC) and managed entities (IO devices). These are only recommendations for Level M3 based servers that apply to all PCIe devices in the system (including Network and Storage Controllers) that support MCTP/PLDM management.

I PLDM [3] [27] [28] [29] is used for the purpose of supporting platform-level data models and platform functions.

X 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.

I MCTP [4] [30] is used 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.

I PLDM over MCTP binding [31] is used as the format of PLDM over MCTP messages.

I PLDM for Firmware Update [38] is used as the messages and data structures used for enabling PLDM devices firmware inventory and update from the BMC.

I PLDM for Redfish Device Enablement [39] is used as the messages and data structures used for enabling PLDM devices to participate in Redfish-based management.

I In addition, Level M3 based server systems are recommended to standardize NVMe Management Interface support with NVMe Management Messages over MCTP.

X Non-Volatile Memory Express (NVMe-MI) [40] is an optimized register interface, command set, and feature set for managing PCIe based NVMe storage. 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 [41] defines how NVMe Management Interface Commands are encapsulated in MCTP Messages and transferred between MCTP Endpoints over the specified transports.

$R_{M3\_IO\_2}$ For Level M3 based server systems, if MCTP/PLDM based management of IO devices is implemented, then the following requirements apply. The physical and data-link layer methods for MCTP communication are minimally defined by the MCTP over SMBus/I2C binding specification [36]. Implementations may choose a higher bandwidth physical data-link, such as MCTP over PCIe VDM [14] or MCTP over I3C [37]

I For BMC-IO devices, SPDM over MCTP support is optional for Level M3 based servers. Conditional requirements for device measurement and authentication are the same as those for BMC and side-band devices.

U Section G provides the use cases when BMC and IO devices communication should be secured with SPDM.

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

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

$R_{M3\_OOB\_1}$ For Level M3-based server systems, the IPMI out of band interface is not required. It is an implementation choice whether IPMI out-of-band is supported or not, and if supported, whether it is enabled or disabled by default.

$R_{M3\_OOB\_2}$ Level M3-based server systems must adhere to the following Redfish requirements:

- Must conform to the DMTF Redfish specification [2] version 1.2 or newer.
  - The conformance of the platform should be verified by executing the Redfish Service Validator and the Redfish Service Conformance Check Tool.
- Must conform to the OCP Baseline Hardware Management Redfish Profile v1.0.1 [9]. It is also recommended to conform to the OCP Server Hardware Management Interface Redfish Profile v1.0.0.
  - The conformance of the platform should be verified by executing the Redfish Interop Validator. The Redfish Interop Validator reads a Profile file as input.
  - The OCP profile JSON files are available at https://github.com/opencomputeproject/OCP-Profiles.

S For more information on using the OCP profiles, refer to [42].

2.5.5 SPDM over MCTP for BMC and side-band devices

I For Level M3-based server systems, using the SPDM protocol for communication with side-band devices is recommended but not required. It is an implementation choice whether SPDM over MCTP is supported or not, and if supported, whether it is enabled or disabled by default.

U Section G provides the use cases when BMC and side-band communication should be secured with SPDM. In this context, “side-band devices” refer to any device that intends to communicate with the BMC using SPDM/MCTP. This includes for example the SatMC, as well as IO Devices and Platform Elements.

R The following are conditional requirements for Level M3-based server systems that implement SPDM over MCTP data protocol:

$R_{M3\_SPDM\_1}$ Must conform to the DMTF SPDM specification [32] version 1.1 or newer.

$R_{M3\_SPDM\_2}$ Must conform to the SPDM over MCTP binding specification [33] version 1.0 or newer.

I BMC should query the side-band device for SPDM support as part of the device discovery procedure.

I BMC should use SPDM attestation mechanisms to verify side-band device authenticity.

I BMC should request side-band device measurements using the SPDM protocol for side-band device firmware validity. These measurements may include the device’s mutable or immutable firmware as well as the device’s hardware and firmware configurations. For device measurement verification, the measurements should be compared to the known good values.

I Side band devices that have access to system critical or confidential data should enforce SPDM mutual authentication and validate BMC authenticity.

I BMC and side-band devices may support Secured Messages using SPDM over MCTP binding [35]

I BMC and side-band device may encrypt data using secure messages with SPDM over MCTP (MCTP Type 6 messages) [35] in case the SPDM secure session is established between BMC and side-band device.

2.6 Level M4

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

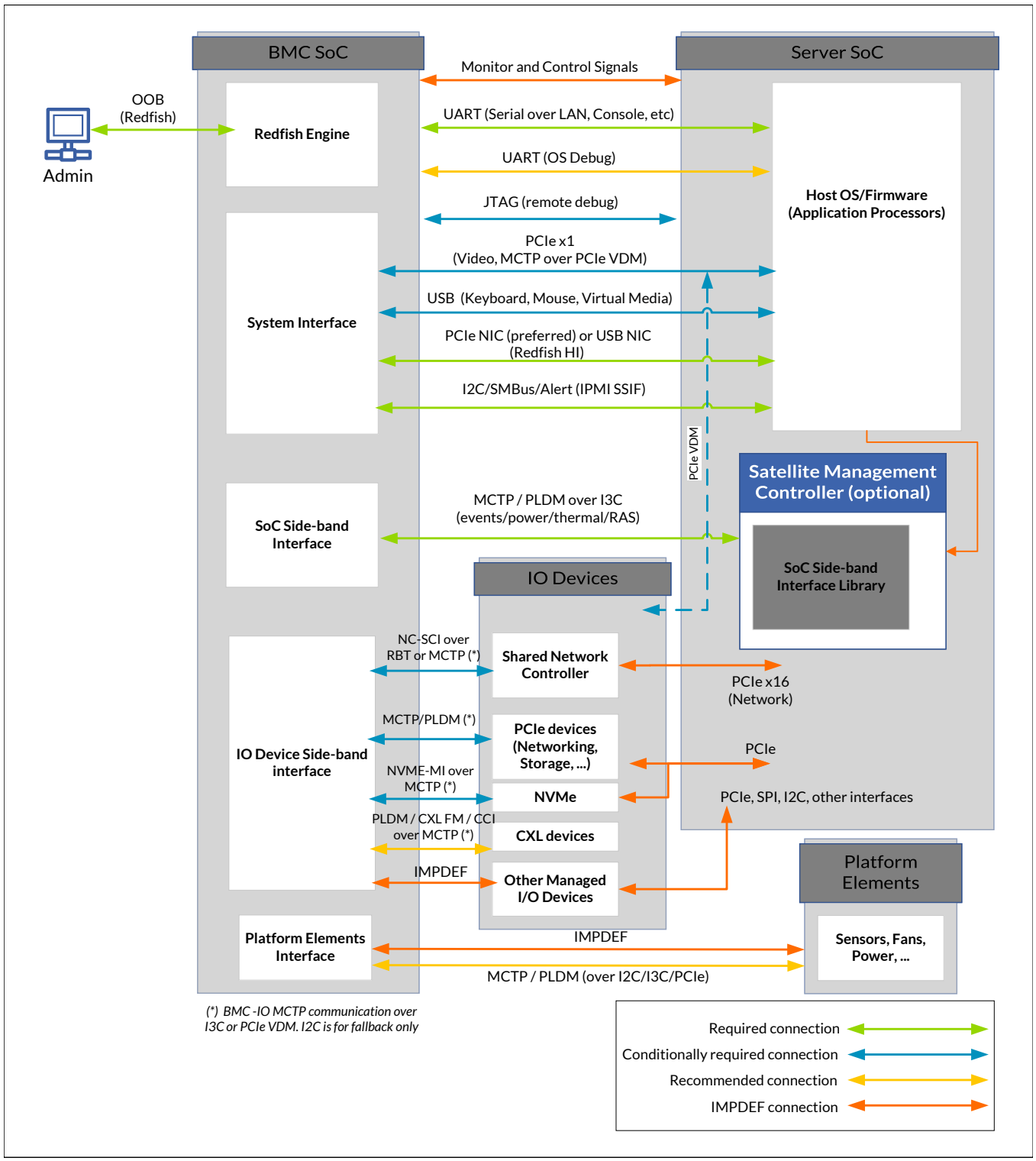

Figure 6: Server Management Interface (Level M4)

2.6.1 SoC-BMC interface

R The requirements and recommendations for these interfaces on Level M4 based server systems are the same as the requirements and recommendations for Level M3 based server systems, with some additional requirements and exceptions.

I For OCP compliant servers with modular design that use a BMC daughter card, SBMR recommends adoption of the Datacenter Secure Control Module 2.0 (DC-SCM)[18]

2.6.1.1 Host SoC in-band interface

$R_{M4\_IB\_1}$ The Redfish Host Interface remains a requirement in Level M4.

I Level M4 servers are recommended to use a PCIe connection to the BMC for communication over PCIe network device, instead of a USB network device.

2.6.1.2 Host SoC side-band interface

$R_{M4\_SB\_1}$ For Level M4 based server systems, the physical and data-link layer methods for MCTP communication are defined by the MCTP over I3C binding specification [37].

2.6.2 BMC-IO device interface

R The requirements and recommendations for these interfaces on Level M4 based server systems are the same as the requirements and recommendations for the Level M3 based server systems, with some additional requirements and exceptions:

$R_{M4\_IO\_1}$ PCIe device management using MCTP/PLDM is a recommendation in Level M3 based server. For Level M4 based servers, this is upgraded to a conditional requirement, applying to all PCIe devices in the system (including Network, Storage Controllers, and NVMe disks) that support MCTP/PLDM management.

$R_{M4\_IO\_2}$ Level M4 based server systems also standardize NVMe Management Interface support with NVMe Management Messages over MCTP. This is a conditional requirement that applies to NVMe devices in the system that support NVMe-MI MCTP management.

I Level M4 based server systems are recommended to standardize CXL devices management using the CXL Fabric Manager API [43] [44].

X The CXL FM API over MCTP Specification [45] defines how the CXL FM API messages are encapsulated in MCTP Messages and transferred between MCTP Endpoints over the specified medium. CXL devices may also support additional management using PLDM over MCTP, similar to other PCIe devices.

I Level M4 based server systems are also recommended to standardize CXL Type 3 device management with CXL Component Command Interface (CCI) messages over MCTP.

X The CXL CCI interface [43] [44] is a register interface and command set for managing CXL Type 3 devices. CXL Type 3 CCI Messages over MCTP Specification [46] defines how the CCI messages are encapsulated in MCTP Messages and transferred between MCTP Endpoints over the specified transport.

$R_{M4\_IO\_3}$ For Level M4 based server systems, the physical and data-link layer methods for MCTP communication are defined by the MCTP over I3C binding specification [37] or over PCIe VDM binding specification [14]. MCTP over SMBus/I2C [36] should be supported only as fallback for older devices that only support MCTP management through I2C.

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

2.6.3 BMC-platform elements interface

I The BMC-Platform Elements interface for the Level M4-based server systems is IMPLEMENTATION DEFINED, with additional recommendations and guidance. Please refer to:

- Intelligent Platform Management Interface v2.0 (IPMI) specification [8].
- OCP server design and specifications [17].
- OCP Datacenter Secure Control Module (DC-SCM) [18].

I In addition, Level M4 based server systems recommend using the DMTF PMCI workgroup standards, when possible.

X These standards define specifications for primary intercommunication interfaces/data models between Management Controller (BMC) and managed entities (Platform Elements). Using these standards for managing platform elements enables advanced functionality such as secure communication, attestation, firmware updates, configuration, and monitoring of the managed entities. For more information refer to [11].

I PLDM [3] [27] [28] [29] is used for the purpose of supporting platform-level data models and platform functions.

X 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.

I MCTP [4] [30] is used 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.

I PLDM over MCTP binding [31] is used as the format of PLDM over MCTP messages.

I SPDM [32] is used for the purpose of supporting security related capabilities of the devices.

X SPDM is designed to provide runtime authentication of a device by retrieving the certificate chains from it and verifying device authenticity by sending unique challenges. SPDM allows the requester to query the device’s firmware or configuration data measurements for device attestation purposes.

I SPDM over MCTP binding [33] is used as the format of SPDM over MCTP messages.

I Secure messages using SPDM specifications [34] is used for the purpose of supporting secure transfer of application data over PMCI transports using SPDM. Secure messages [34] also defines transport requirements for SPDM records, which form the basis of encryption and message authentication.

I Secured Messages using SPDM over MCTP binding [35] is used as the format of SPDM secure messages over MCTP messages.

X 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 [36]. Additional MCTP physical and data-link layers are defined for I3C [37] and PCIe VDM [14].

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

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

R The requirements and recommendations for the BMC out-of-band interfaces for Level M4-based server systems are the same requirements and recommendations as for Level M3-based server systems.

2.6.5 SPDM over MCTP for BMC and side-band devices

R For Level M4-based server systems, the requirements are the same as for M3-based server systems.

2.7 SBMR checklist

This section lists the minimum SBMR server requirements.

2.7.1 SBMR Level M1 checklist

Category	Rule ID
In-Band	M1_IB_1
UART	M1_UART_1
UART	M1_UART_2
JTAG	M1_JTAG_1
JTAG	M1_JTAG_2
OOB	M1_OOB_1
IPMI	IPMI_1
IPMI	IPMI_2
IPMI	IPMI_3
IPMI	IPMI_4
IPMI	IPMI_5
IPMI	IPMI_6
IPMI	IPMI_7

2.7.2 SBMR Level M2 checklist

In addition to the SBMR Level M1 rules in Section 2.7.1, the following additional rules are required.

Category	Rule ID
In-Band	M2_IB_1
In-Band	M2_IB_2
JTAG	M2_JTAG_1
JTAG	M2_JTAG_2
BMC-IO	M2_IO_1
OOB	M2_OOB_1
OOB	M2_OOB_2
OOB	M2_OOB_3

2.7.3 SBMR Level M2.1 checklist

In addition to the SBMR Level M2.1 rules in Section 2.7.2, the following additional rules are required.

Category	Rule ID
In-Band	M21_IB_1
In-Band	M21_IB_2
In-Band	M21_IB_3
PCIe	M21_PCI_1
USB	M21_USB_1
IPMI	M21_IPMI_1
IPMI	M21_IPMI_2
RAS	M1_RAS_1
RAS	M1_RAS_2
RAS	M2_RAS_1
RAS	M2_RAS_2

2.7.4 SBMR Level M3 checklist

In addition to the SBMR Level M3 rules in Section 2.7.3, the following additional rules are required.

Category	Rule ID
Side-Band	M3_SB_1
Side-Band	M3_SB_2
Side-Band	M3_SB_3
Side-Band	M3_SB_4
Side-Band	M3_SB_5
Side-Band	M3_SB_6
Side-Band	M3_SB_7
Side-Band	M3_SB_8
Side-Band	M3_SB_9
JTAG	M3_JTAG_1
JTAG	M3_JTAG_2
BMC-IO	M3_IO_1
BMC-IO	M3_IO_2
OOB	M3_OOB_1
OOB	M3_OOB_2
SPDM	M3_SPDM_1

Category	Rule ID
SPDM	M3_SPDM_2

2.7.5 SBMR Level M4 checklist

In addition to the SBMR Level M4 rules in Section 2.7.4, the following additional rules are required.

Category	Rule ID
In-Band	M4_IB_1
Side-Band	M4_SB_1
BMC-IO	M4_IO_1
BMC-IO	M4_IO_2
BMC-IO	M4_IO_3

A OpenBMC

The OpenBMC project 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. The project Technical Steering Committee includes Facebook, Google, IBM, Intel, and Microsoft, a well as Arm.

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

B IPMI

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

R_{IPMI_1} A platform must provide a mechanism for remotely powering an individual node on and initiating the boot sequence.

B.1.1.2 Power off

R_{IPMI_2} 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

R_{IPMI_3} 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

R_{IPMI_4} 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

R_{IPMI_5} 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

R_{IPMI_6} 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

R_{IPMI_7} 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

I 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 [22].

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

I `CredentialBootstrapping` property defined in the `HostInterface` Redfish Schema [7] [47]. Platforms should implement this property as a writeable configuration setting to allow the administrator to disable the bootstrapping facility for security reasons.

I `CredentialBootstrappingRole` property in the Links property defined in the HostInterface Redfish Schema [7] [47].

B.1.6 IPMI support verification

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

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

Table 10 lists Arm standard IPMI commands that are defined in SBMR.

Table 10: List of Arm standard IPMI commands

Command	NetFn	Command Code	Definition
Send Platform Error Record	2Ch	01h	Section C.2.2
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 in Table 11 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.

Table 11: SBMR deviations from the IPMI specification

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**."

IPMI §	Existing language	Updated language
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”
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 Table 13:

Table 13: IPMI SSIF Multi-part Start transactions

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 Table 14:

Table 14: IPMI SSIF Multi-part Middle transactions

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 Table 15:

Table 15: IPMI SSIF Multi-part End transactions

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	

U 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

This section covers requirements and guidance for handling and implementing platform Reliability, Availability and Serviceability (RAS) events.

C.1 Level M0

I 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

I Figure 7 shows a conceptual illustration of IPMI based in-band, SoC side-band, and out-of-band RAS interfaces for Level M1-based server systems.

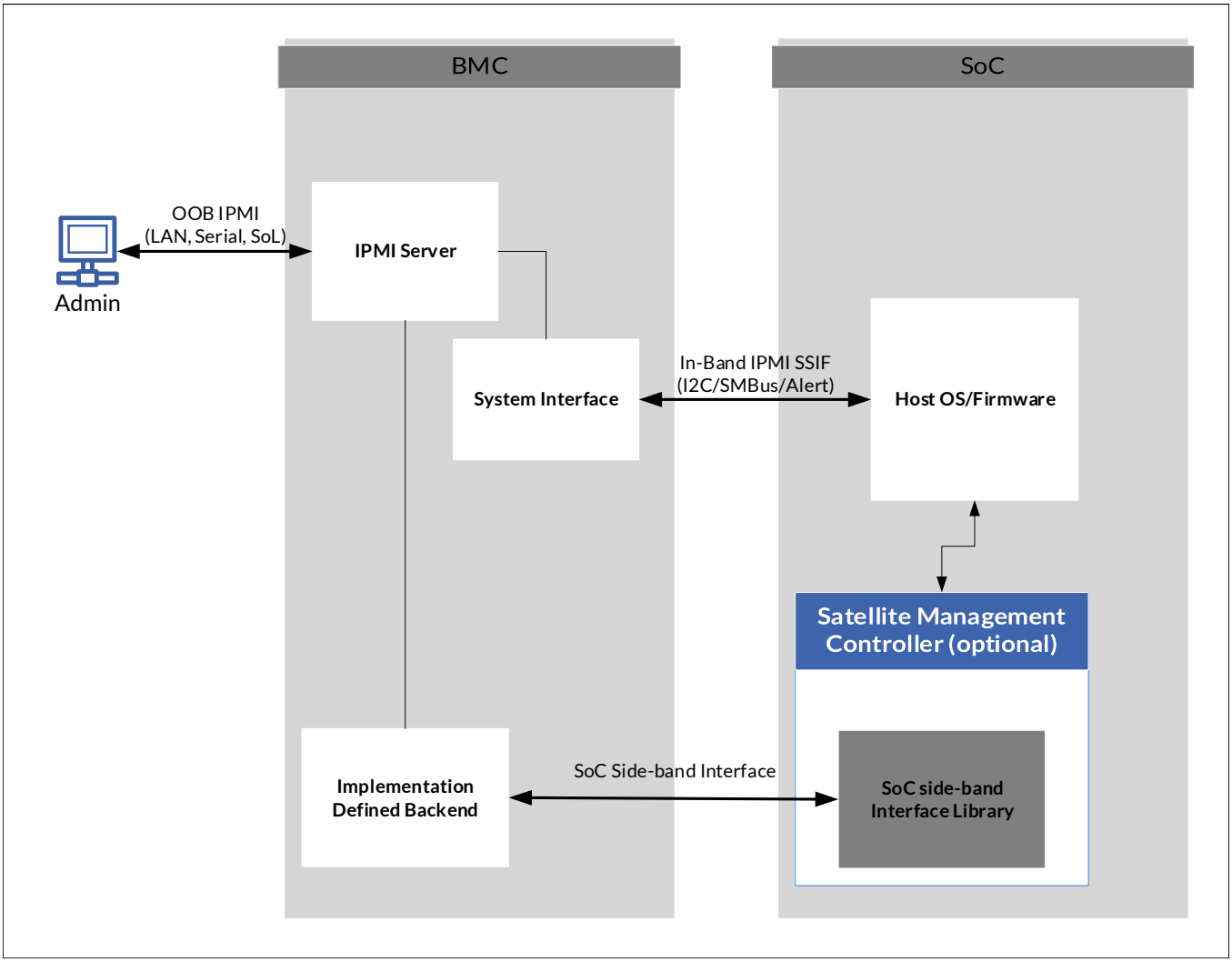

Figure 7: IPMI based RAS Interfaces

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

I 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. The SSIF interface is intended to be used by host OS and firmware to communicate with the BMC. Once the host boots to the OS, this interface is typically used only by the OS.

I The format of the IPMI command used over this interface to send the RAS platform errors to the BMC is defined in section Section C.2.2.

I 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.

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

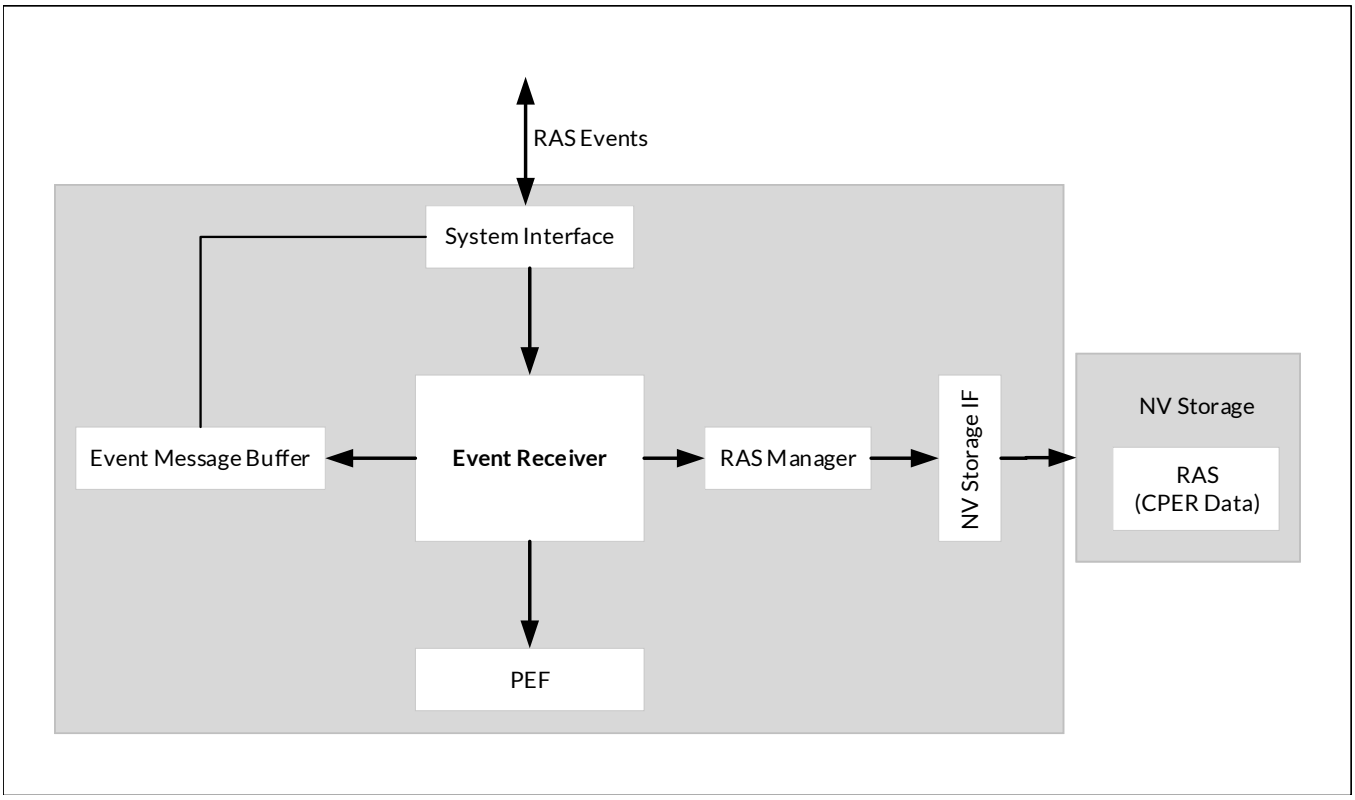

Figure 8: IPMI based RAS Event Receiver

U 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 (CPER) entry. Finally, the RAS Manager function calls calls the Non-Volatile Storage Interface to store the event record.

$R_{M1\_RAS\_1}$ SBMR requires the error record data format to be in raw Common Platform Error Record (CPER) format when using this interface. The format of CPER is defined in UEFI Specification [6] (UEFI § N). When creating CPER raw files for logging to or extracting from the BMC, SBMR requires that the CPER data contain a single CPER Section (Section Descriptor and Section), as defined by the UEFI Specification Appendix N.2.2.

C.2.2 RAS IPMI message format

$R_{M1\_RAS\_2}$ 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 (Section Descriptor and Section)

Response Data

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

I 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

I 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

I 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

$R_{M2\_RAS\_1}$ 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.

I Figure 9 shows a conceptual illustration of these interfaces for RAS.

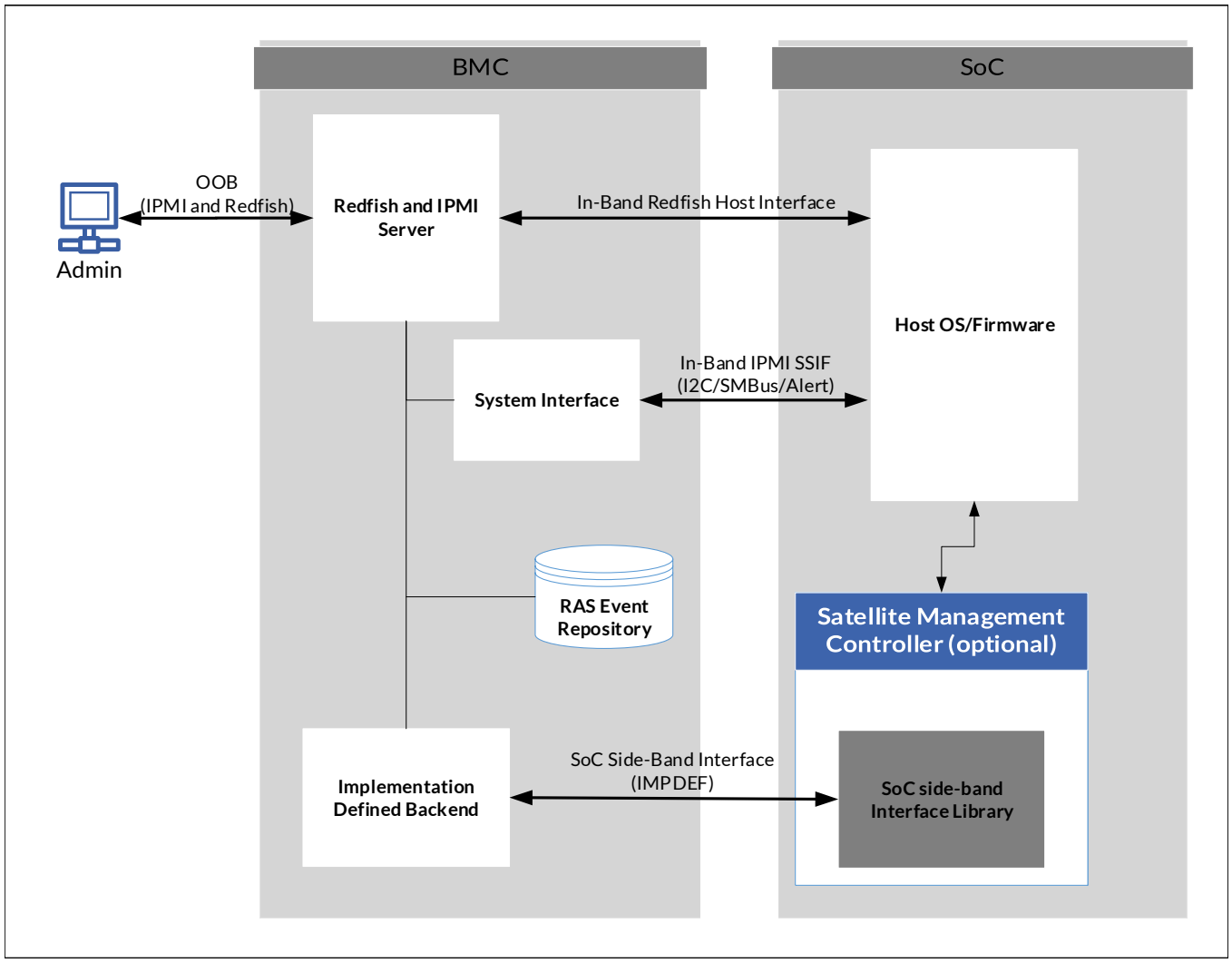

Figure 9: Redfish and IPMI based RAS Interfaces

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

I 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.

I Arm recommends storing the error records in CPER-like format in the RAS Event Repository non-volatile storage.

I 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

$R_{M2\_RAS\_2}$ The Redfish model for extracting Platform Error Records is defined in DMTF Redfish Schema Supplement [42], under the LogEntry 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, as defined by Redfish LogEntry v.10.0 schema, and SBMR section C.2
- `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`, `CPER`, `CPERSection`, or `OEM`.
  - When using Redfish to report CPER data, SBMR requires setting this property to `CPER` when the content is a complete CPER Record with a Header and one or more Sections, or `CPERSection` when the content is a single CPER Section (and a Section Descriptor) without a Header. This allows user software to distinguish CPER records from other diagnostics files.

C.3.3 SoC side-band interface

I 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

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

C.4 Level M3 and M4

$R_{M3\_RAS\_1}$ 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.

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

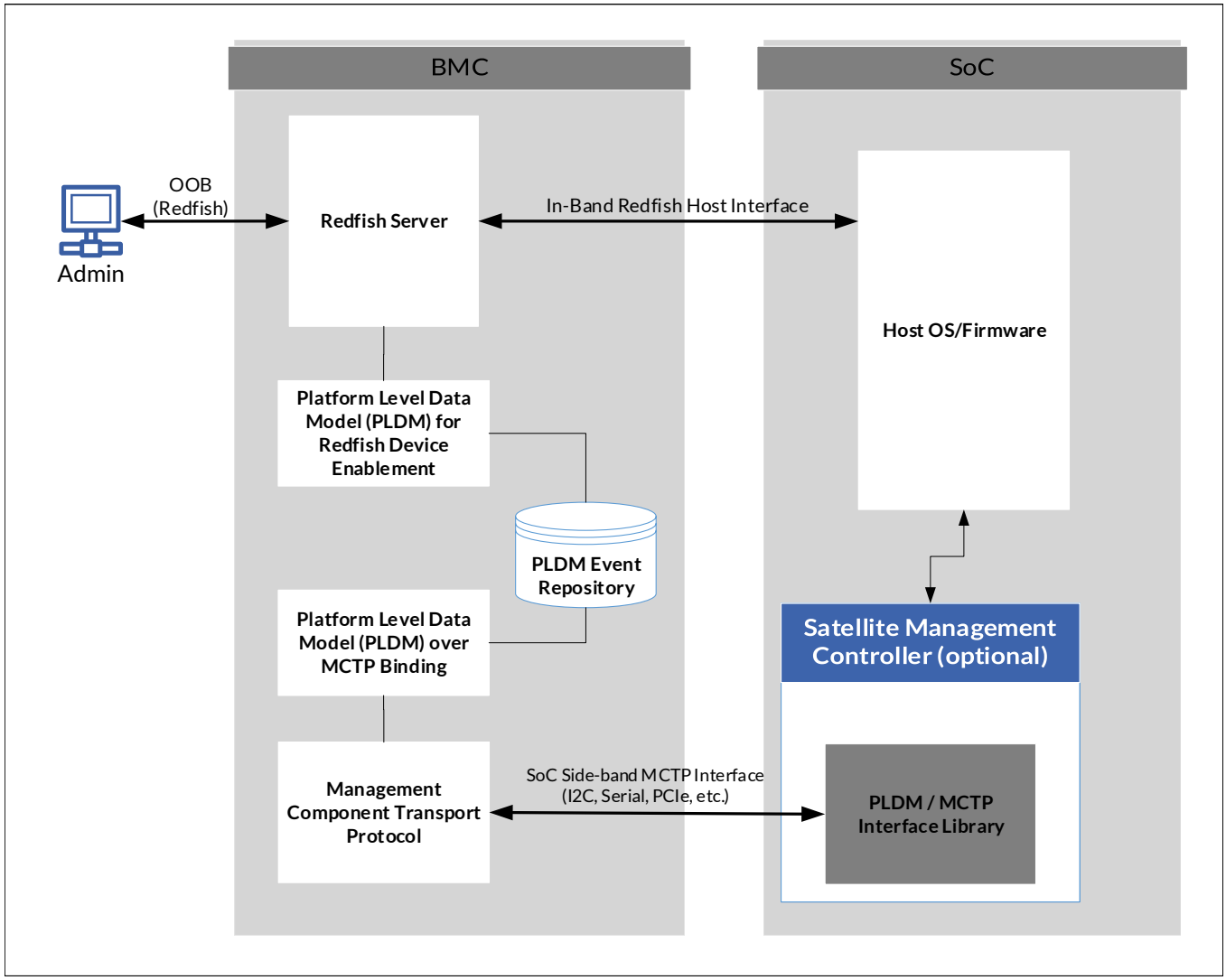

Figure 10: Redfish/PLDM/MCTP based RAS Interfaces

C.4.1 Redfish host (in-band) interface

I 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 and PLDM (SoC side-band) interface

I 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 M3-based and M4-based server systems.

I SBMR recommends Platform Level Data Model (PLDM) as the SoC side-band message definition and data layer interface for the Level M3/M4 based server systems.

I SBMR recommends that the error record data format is in CPER format when using this interface.

I SBMR recommends the use of `PlatformEventMessage`, `PollForPlatformEventMessage`, `EventMessageSupported` ↪ and `EventMessageBufferSize` Commands to transfer CPER formatted RAS errors from the Satellite Management Controller to the BMC.

I A new event class `CPEREvent` is proposed to enable this feature:

- SBMR defines `oemEvent` of value **FA** for the `CPEREvent` event class, until a new standard value is defined in [28] Table 11. The format for `CPEREvent` is defined in Table 18.
- In addition, SBMR recommends the BMC use of `pldmMessagePollEvent` to allow for asynchronous polling of error event, as well as the transfer of large `CPEREvent` messages.

U When the BMC receives a `pldmMessagePollEvent`, it is a signal that event FIFO contains a large message that will require multipart transfers. The BMC then uses the `PollForPlatformEventMessage` command with `TransferOperationFlag` set to `GeNextPart` 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 event message. 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`. Finally, the BMC can verify if `eventClass` field of re-assembled event message is `CPEREvent`.

For more details, refer to [28].

C.4.3 RAS PLDM message format

The proposed `CPEREvent` event data format for RAS/CPER PLDM event log is shown in Table 18.

Table 18: CPEREvent eventData format

Type	Request data
uint8	`formatVersion` Version of event format (the format and definition of the following bytes) - 0x01 for this specification
uint8	`formatType` The type of Error in eventData - 0x00 = Common Platform Error Record (CPER) – Full Record with Header and one or more Sections - 0x01= Single CPER Section - 0x02. . . 0xFF = Reserved
uint16	`eventDataLength` Length in bytes of the eventData field below
var	`eventData` • Format type = 0x00 A chunk of CPER formatted data including record header, one or more section descriptors, and one or more sections, as described in UEFI specification [6] appendix N • Format type = 0x01 A chunk of CPER formatted data that contains a single section descriptor and section, without the header, as described in UEFI specification [6] appendix N

C.4.3.1 RAS PLDM message flows examples

U Figure 11 shows an example flow when BMC and SatMC boot up to exchange the capabilities, such as max buffer size, supported event types, asynchronous and polling mode.

U Figure 12 shows an example flow that SatMC use asynchronous method to send small event to BMC.

U Figure 13 shows an example flow that SatMC use asynchronous method to notify BMC to switch to polled event transfer to receive a large multi-part event.

U Figure 14 shows an example flow that BMC use polling method to receive a large multi-part event.

U Figure 15 shows an example flow that BMC use polling method to receive a small event.

U Figure 16 shows an example flow that SatMC reports an empty event queue when BMC try to poll an event back.

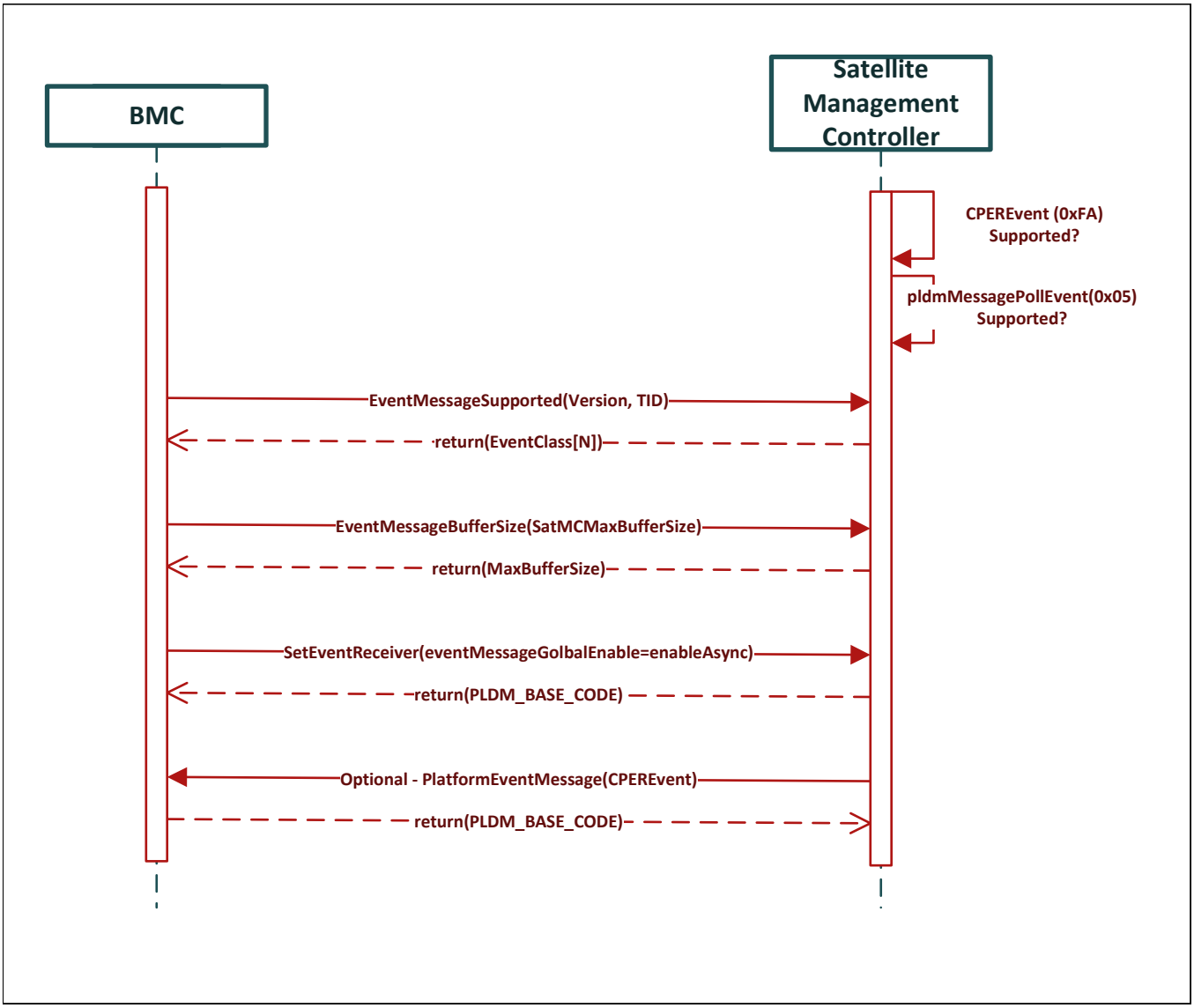

Figure 11: RAS side-band flow – boot initialization

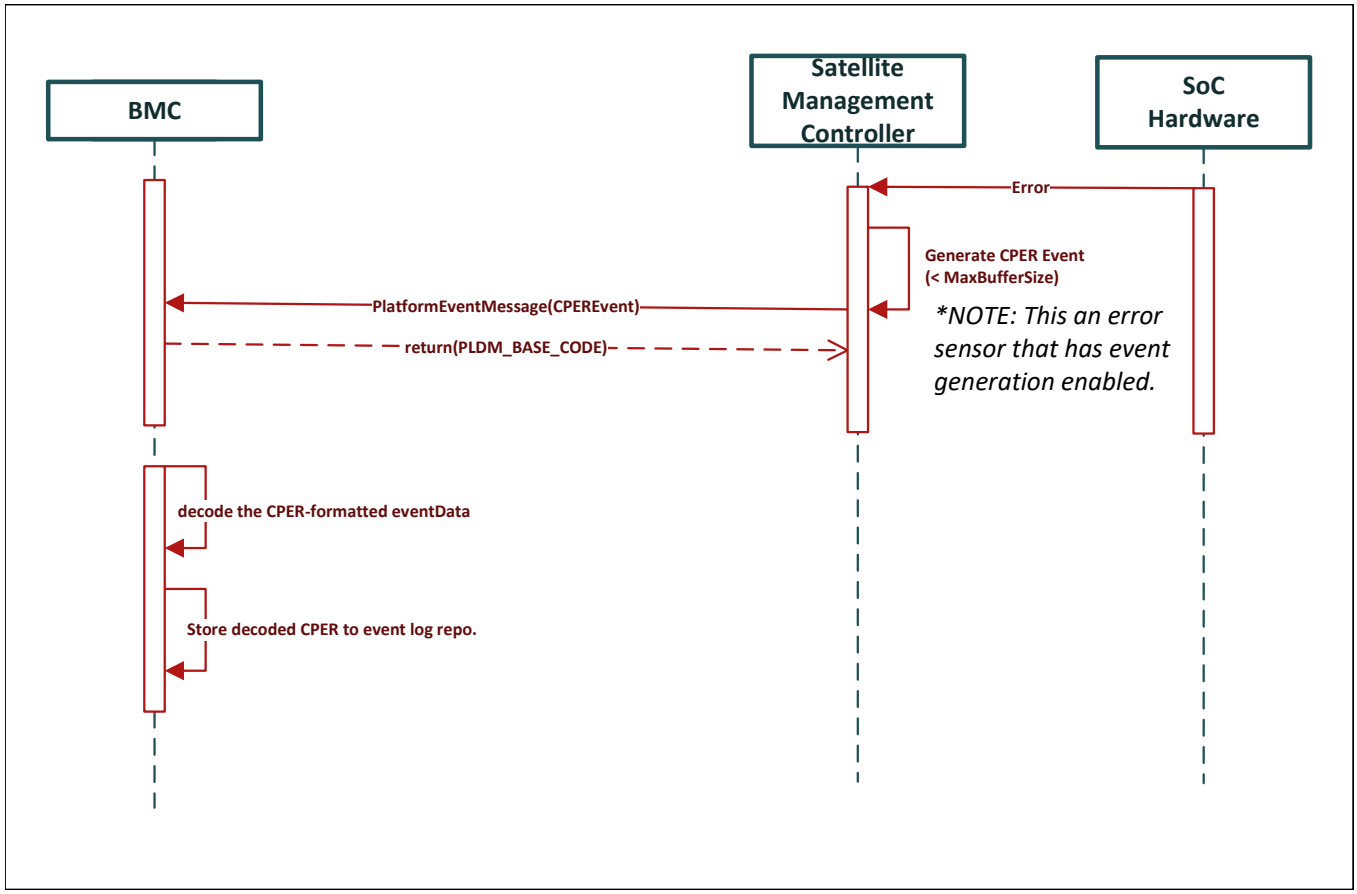

Figure 12: RAS side-band flow – Asynchronous small event

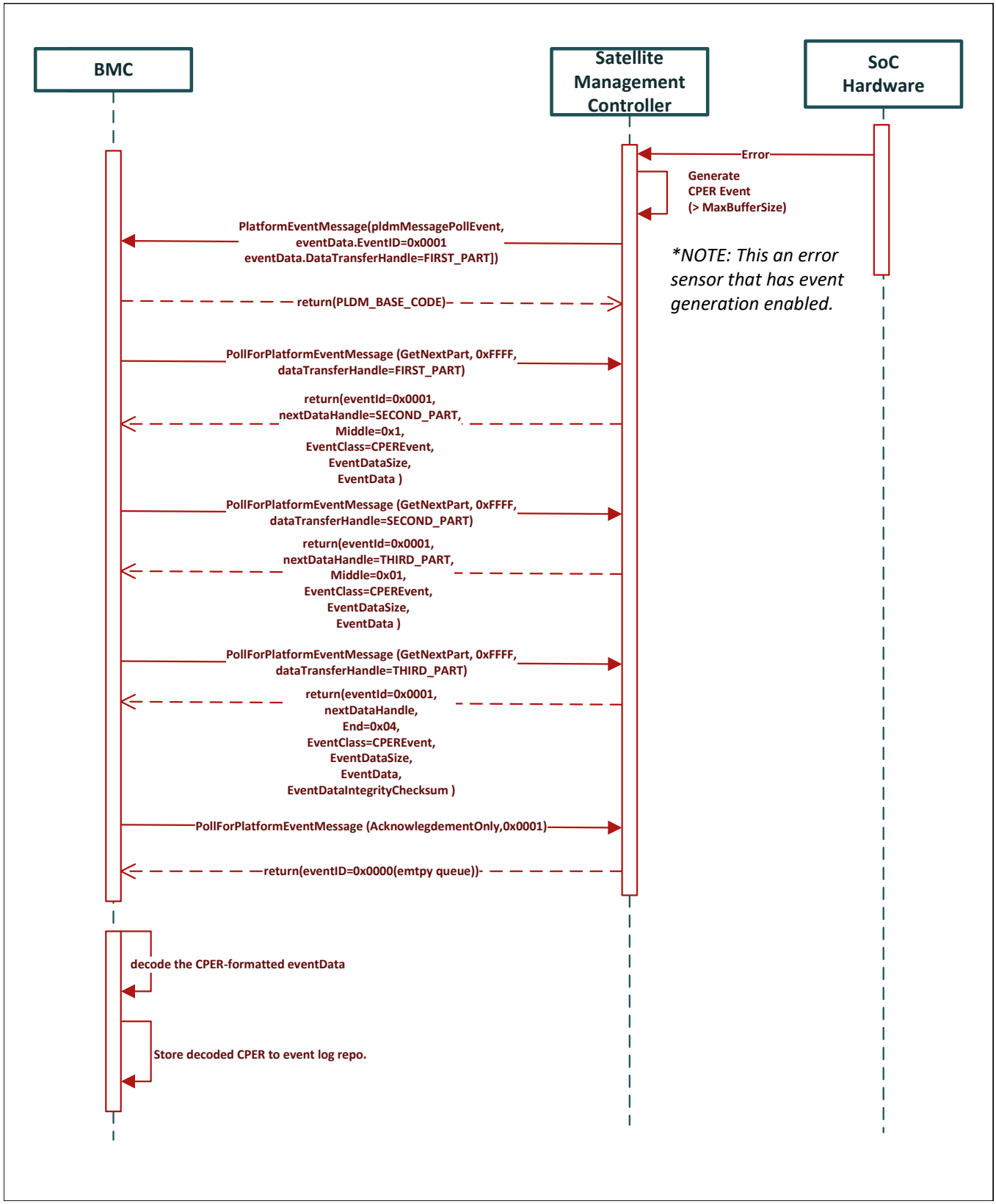

Figure 13: RAS side-band flow – SatMC async notification for BMC to switch to poll for large event

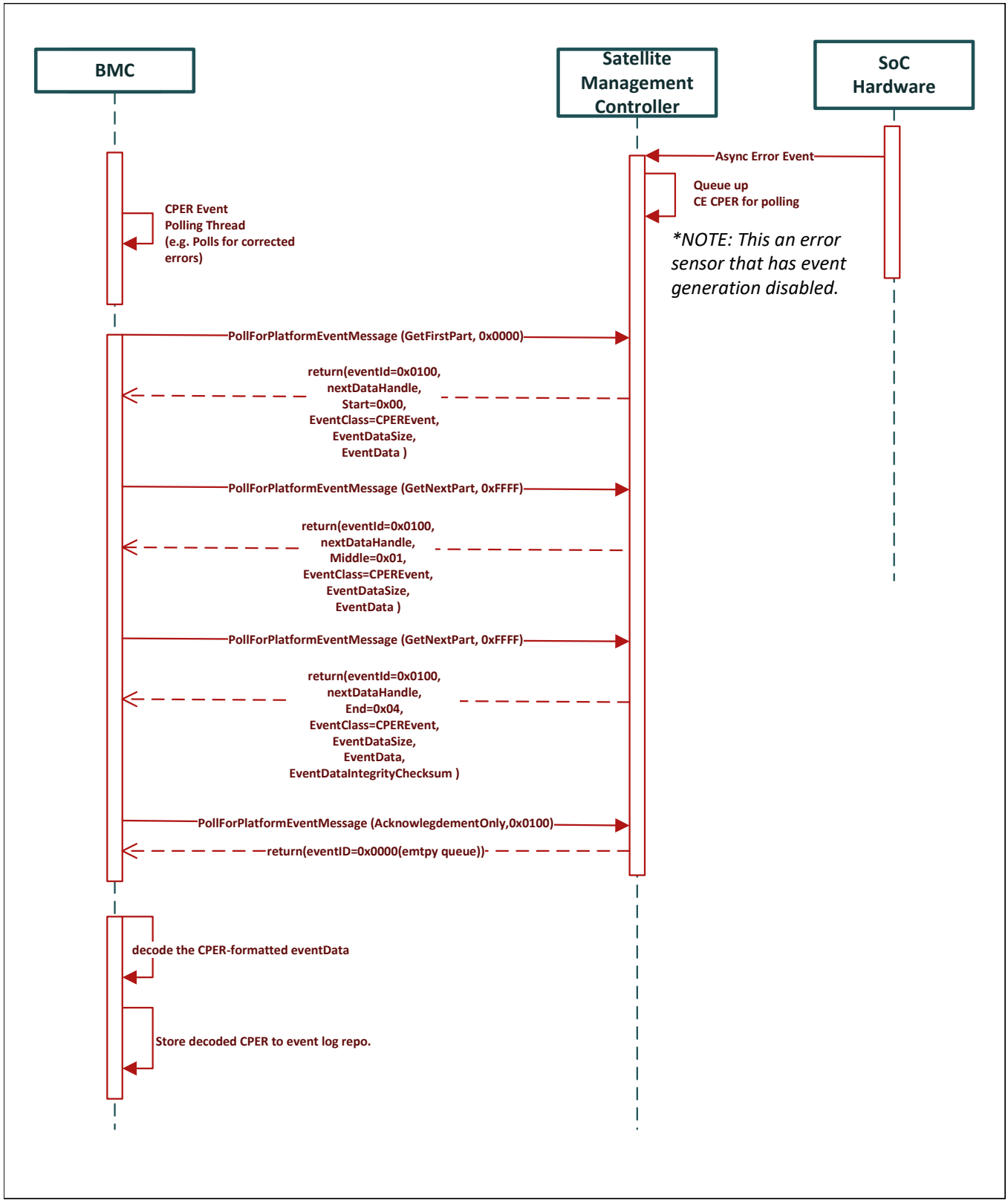

Figure 14: RAS side-band flow – BMC Polling to receive large event

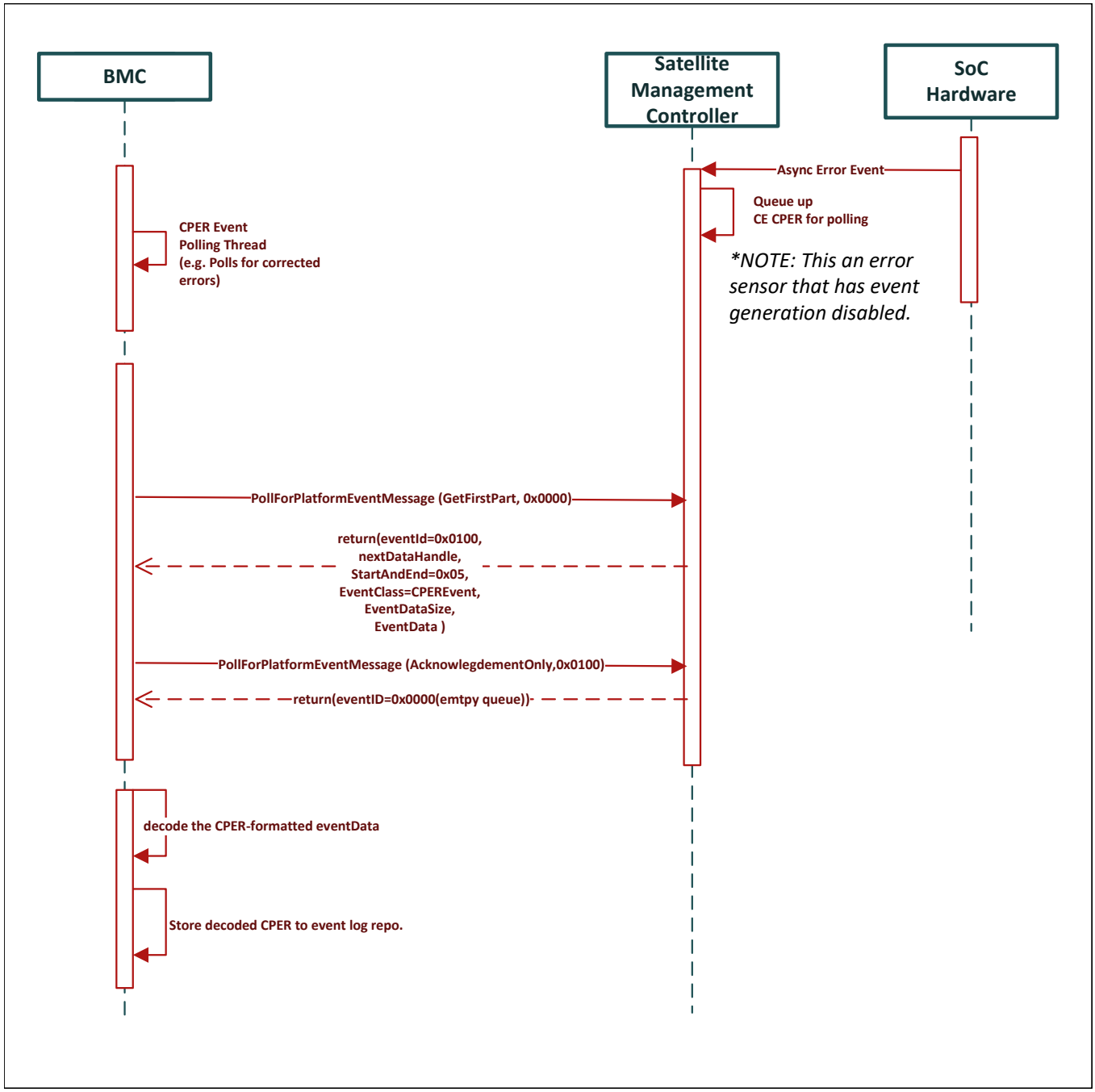

Figure 15: RAS side-band flow – BMC Polling to receive small event

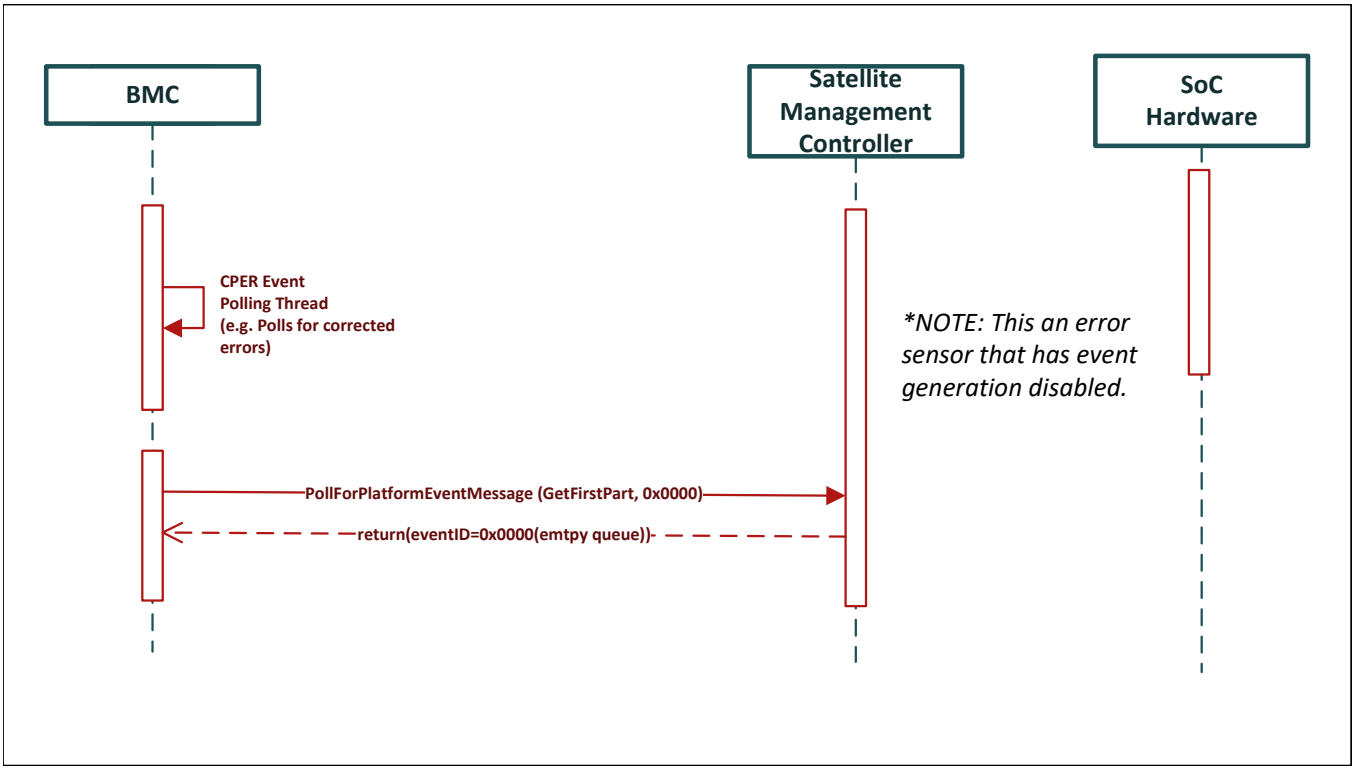

Figure 16: RAS side-band flow – SatMC report empty event queue

C.4.4 Out-of-band interface

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

U When the BMC polls the CPER binary from the SatMC successfully and stores it in its local repository, the BMC generates corresponding event log entry to Redfish event log repository as explained in Section C.3.2. A Redfish based tool can then retrieve the CPER binary data from BMC by iterating through the Redfish log entries, looking for a Log entry with LogDiagnosticDataType set to CPER or CPERSection. Once found, the tool can download the CPER file pointed to by the AdditionalDataURI property of the Redfish log entry for that error event.

Figure 17 shows a conceptual illustration of the CPER dataflow from Satellite Management Controller to Remote Redfish base tool.

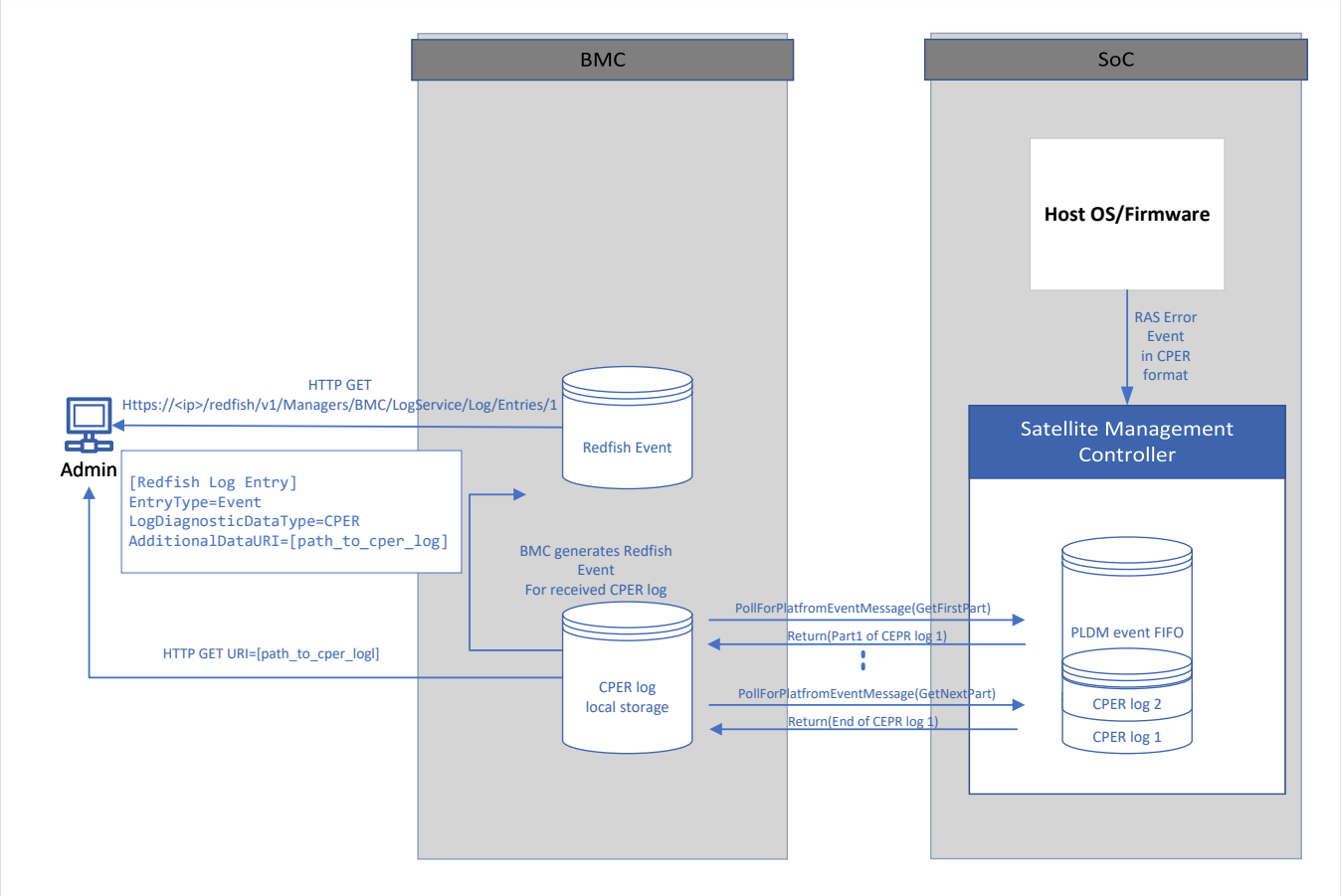

Figure 17: Out-of-band interface for RAS/CPER event log

D Platform Monitoring and Control

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

I 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)

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

I 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

X 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

I 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

I SBMR recommends the use of Entity IDs which identify the sensor association with a physical container. SBMR reserves the following Entity IDs in Table 19 to identify SoC firmware (e.g. pre-EFI firmware), and SoC Management Software (e.g. Satellite/Service Management Software). These values are reserved from the OEM System Integrator defined range 0xD0 – 0xFF.

Table 19: IPMI Entity IDs

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

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

D.4 PLDM commands/APIs to monitor and control managed entities

I SBMR recommends that the SatMC supports the following list of PLDM commands in Table 20 and Table 21 for monitoring and control of SoC-connected Numeric and State managed entities/effecters:

Note

The “M”, “C”, “O” below stand for Mandatory, Conditional, and Optional, respectively.

Table 20: PLDM platform commands

PLDM Platform command	M/C/O	Responder	Description
SetNumericSensorEnable	C	SatMC	To be implemented when SatMC
GetSensorReading	C	SatMC	has numeric sensor(s)
SetSensorThresholds	O	SatMC	
SetStateSensorEnables	C	SatMC	To be implemented when SatMC
GetStateSensorReadings	C	SatMC	has state sensor(s)

PLDM Platform command	M/C/O	Responder	Description
SetNumericEffecterEnable	C	SatMC	To be implemented when SatMC
SetNumericEffecterValue	C	SatMC	has numeric effecter(s)
GetNumericEffecterValue	O	SatMC	
SetStateEffecterEnables	C	SatMC	To be implemented when SatMC
SetStateEffecterStates	C	SatMC	has state effecte(s)
GetStateEffecterStates	O	SatMC	
SetTID	C	SatMC	To be implemented when SatMC
GetTID	O	SatMC	supports event message logging.
SetEventReceiver	C	SatMC	
GetEventReceiver	O	SatMC	
PlatformEventMessage	C	BMC	
PollForPlatformEventMessage	C	SatMC	To be implemented when SatMC needs to log large event message.

Table 21: PLDM FRU commands

PLDM FRU command	M/C/O	Responder	Description
GetFRURecordTableMetadata	M	SatMC	BMC uses the command to check if SatMC has FRU data available
GetFRURecordTable	M	SatMC	BMC uses the command to get FRU data of SatMC back.

Platform Descriptor Records (PDRs)

X Platform Descriptor Records (PDRs) provide semantic information for managed entities. PDRs are optional for PLDM-based platform monitoring, and whether they are used or not depends on the PLDM sub-system implementation. It is possible to support PLDM-based platform monitoring using PLDM-only accesses, or using PLDM with Device PDRs, as explained in PLDM for Platform Monitoring and Control Specification § 8.3 [28]

I If PDRs are used, SBMR recommends the following list of PLDM commands in Table 22 for management of PDRs of managed entities:

Table 22: PLDM PDR FRU commands

PLDM FRU command	M/C/O	Responder	Description
GetPDRReositoryInfo	C	SatMC	If PDRs are used, then SatMC must implement this command if PDRs are used. This is needed for BMC to check if any PDR is available.

PLDM FRU command	M/C/O	Responder	Description
GetPDR	C	SatMC	If PDRs are used, then SatMC must implement this command if PDRs are used. This is needed for the BMC to fetch PDRs to identify the SatMC.
RunInitAgent	O	BMC	The command is optional depending on implementation. It will be useful to have another management controller, system firmware, or another entity to trigger the PLDM initialization process.

For more details on the PLDM Commands, refer to [28].

I If PDRs are used, SBMR recommends the following types of Platform Descriptor Records (PDRs) in Table 23 to be implemented.

Table 23: PLDM PDR types

PDR type	M/C/O	Responder	Description
FRU Record Set PDR	C	SatMC	If PDRs are used, then SatMC must implement this PDR, with the Entity Type Field value set to 0x2E (Management Controller Firmware) for BMC to identify itself.
Terminus Locator PDR	C	BMC	If PDRs are used, then SatMC must implement this PDR. BMC needs to update the PDR when there is new SatMC added to system. The event log viewer needs the data to identify where the event messages are originating from.
Numeric Sensor PDR	O	SatMC	
Numeric Sensor Initialization PDR	O	SatMC	If PDRs are used, SatMC should implement this PDRs if SatMC supports Numeric Sensor(s) and BMC has no knowledge of accessing this SatMC.
State Sensor PDR	O	SatMC	If PDRs are used, SatMC should implement
State Sensor Initialization PDR	O	SatMC	this PDRs if SatMC supports State Sensor(s) and BMC has no knowledge of accessing these sensors.
Numeric Effecter PDR	O	SatMC	If PDRs are used, SatMC should implement this
Numeric Effecter Initialization PDR	O	SatMC	PDRs if SatMC supports Numeric Effecter(s) and BMC has no knowledge of accessing these sensors.
State Effecter PDR	O	SatMC	If PDRs are used, SatMC should implement this
State Effecter Initialization PDR	O	SatMC	PDRs if SatMC supports State Effecter(s) and BMC has no knowledge of accessing these sensors.

U The flowchart in Figure 18 demonstrates how the BMC uses PLDM commands and PDRs to retrieve sensor / platform monitoring data from a Satellite Management Controller.

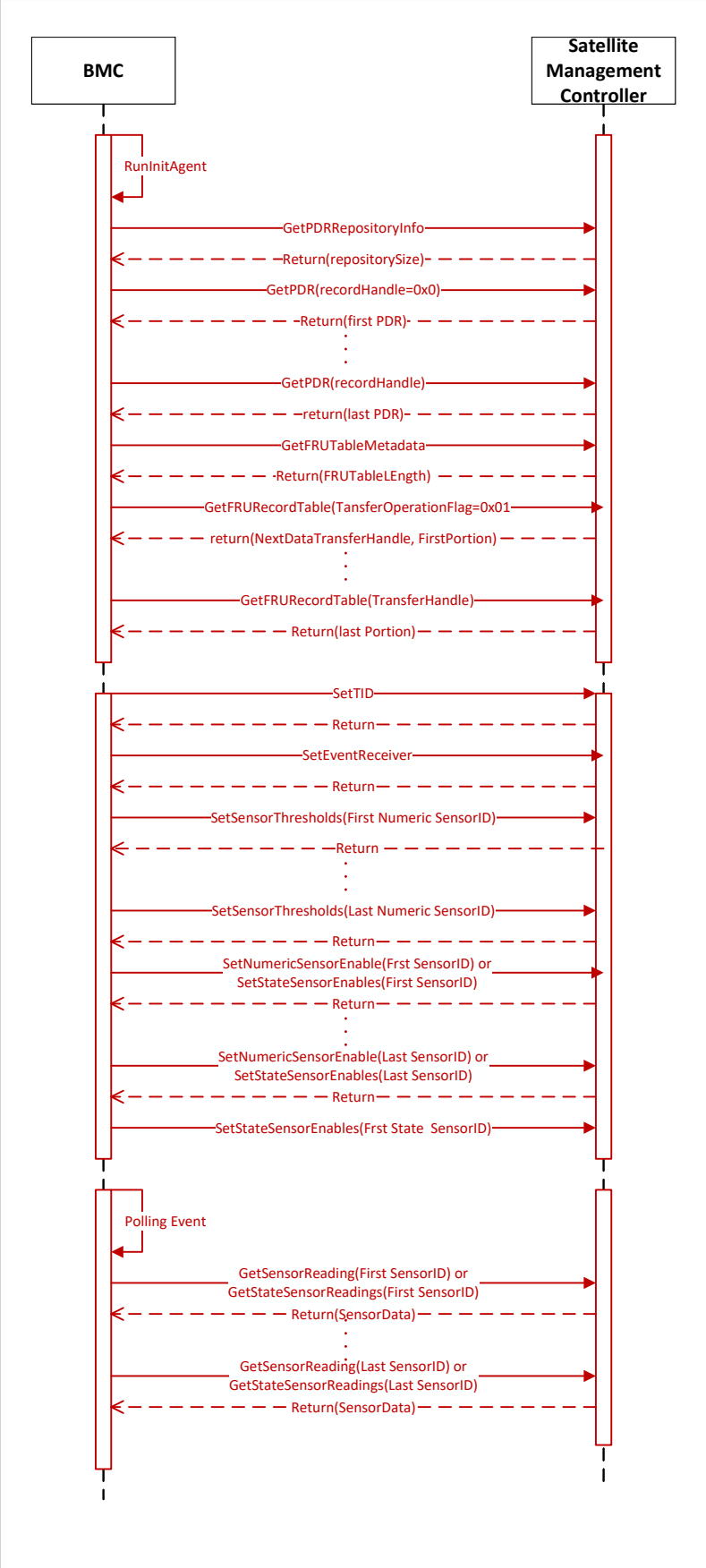

Figure 18: BMC PLDM/PDR monitor from SatMC

D.4.1 Examples of PLDM sensors exposed by SatMC

I The types and numbers count of sensors exposed by SatMC is IMPLEMENTATION DEFINED. SBMR recommends that the SatMC utilizes Set ID/Entity ID codes that are defined in [29] whenever possible.

U The following table Table 24 shows examples of typical sensors that would exist in a server SoC.

Table 24: Examples of PLDM sensors

Name	Sensor Type	Entity Type	Set ID	Description
CPU temp.	Numeric	135, Processor	N/A	Needed by BMC thermal management. The type/number of temp sensors is IMPLEMENTATION DEFINED
CPU Power State	State	135, Proccessor	288, Proccesor Power Sate	The Set ID value of the state sensor PDR should be 288 if the ACPI power state in DSP0249[29] table 10 can be applied to theCPU/SoC in system
CPU Power Meter	Numeric	135, Processor	N/A	The Numeric sensor shows the current power consumption of CPU/SoC. The sensor can be implemented if SatMC has ability to measure the current of CPU/SoC.
CPU Performance Level	State	135, Processor	289, Power-Performance State	The Set ID of state sensor PDR should be value 289 if the ACPI power state in DSP0249[29] table 10 can be applied to the CPU/SoC in system.
DIMM Group N max. temp	Numeric	66, Memory module	N/A	1. Reporting the hottest temperature in the DIMM group for BMC thermal management 2. To be implemented when SatMC can access to the SPD of DIMM. 3. The number of DIMM group depends on CPU/SoC design. It could be 1 or many.
DIMM Group N Power Meter	State	66, Memory Module	N/A	1. The Numeric sensor shows the current power consumption of DIMM group N. 2. It can be implemented if SatMC has | ability to measure the power consumption of DIMMs in system. 3. The number of DIMM group depends on CPU/SoC design. It could be 1 or many.

E Reference Implementation of Remote Debug 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, M3, M4

U This section describes a reference solution for implementing BMC remote debug using OpenOCD for SBMR Levels M1, M2, M2.1, M4, and M4 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.

S 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.

S 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)

U A reference implementation of remote debug feature using GNU MCU Eclipse plugin, OpenOCD using JTAG interface is shown in Figure 19.

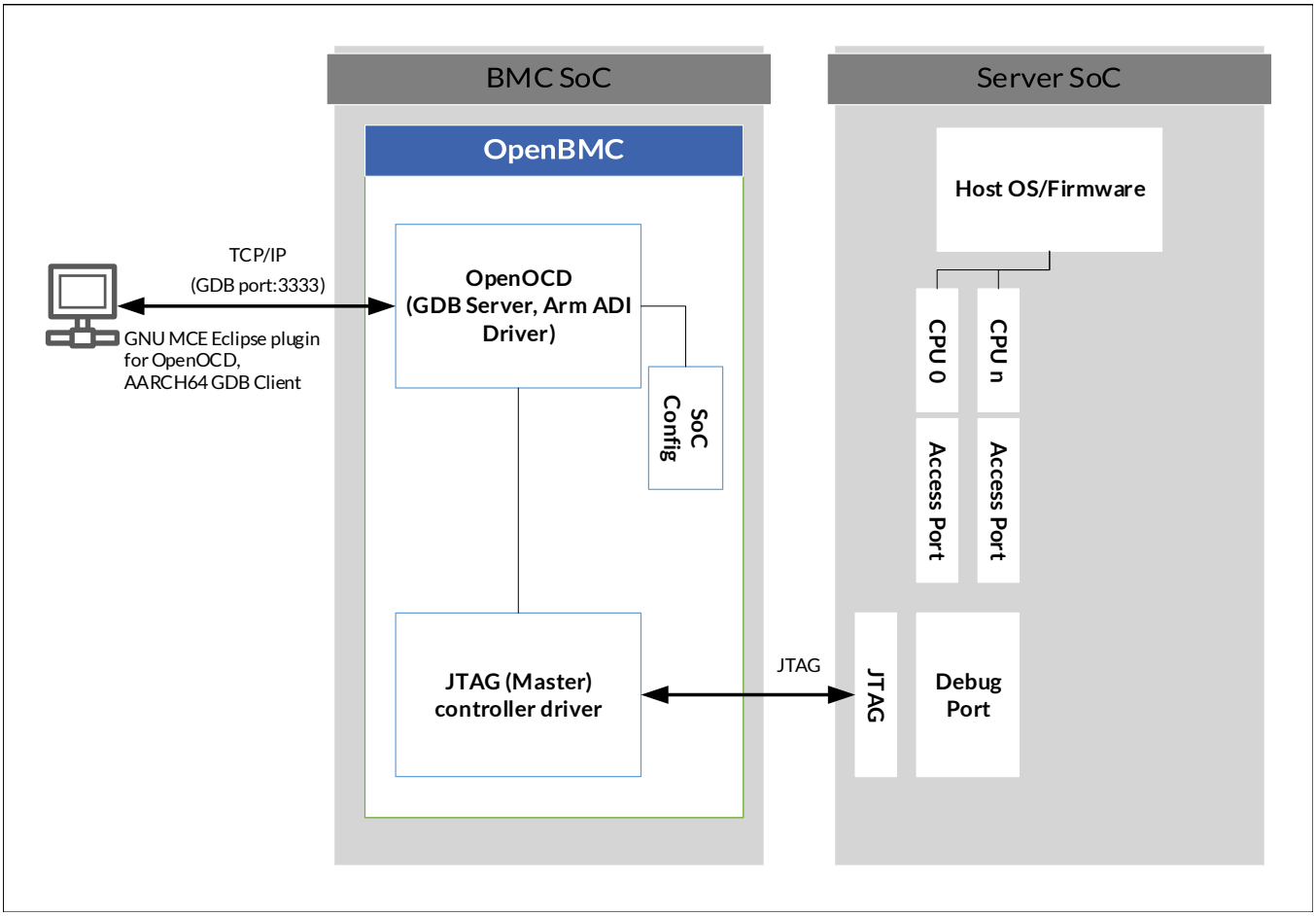

Figure 19: Reference implementation of remote debug.

U 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.

U 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 [48].

U 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 Codes

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)

I 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.

U 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 [49]. 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. The format is specified in Table 29 below:

Table 29: SBMR Boot Progress Codes format

Byte offset	Size (Bytes)	Description	Details
0	1	`STATUS_CODE_TYPE 0x01 = PROGRESS_CODE 0x02 = ERROR_CODE 0x03 = DEBUG_CODE`	32-bit field that follows the format of `EFI_STATUS_CODE_TYPE` as defined by the PI Specification [49] (PI § Vol 1-4.7 PI § Vol 2-14.2, PI § Vol 3- 6).
1	2	`STATUS_CODE_RESERVED Reserved by PI Specification. set to 0x0000`	
3	1	`STATUS_CODE_SEVERITY 0x40 = ERROR_MINOR 0x80 = ERROR_MAJOR 0x90 = ERROR_UNRECOVERED 0xa0 = ERROR_UNCONTAINED`	
4	2	`EFI_STATUS_CODE_OPERATION 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 `EFI_STATUS_CODE_VALUE` as defined by the PI Specification [49] (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 [49] (PI § Vol 1-4.7 PI § Vol 2-14.2, PI § Vol 3- 6).

F.2.1 Example progress codes (IPMI)

U 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 ERROR_UNRECOVERED = 0x90	0x90000002
	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 ERROR_UNRECOVERED = 0x90	0x90000002
	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] [47] 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.

I 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.

U 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

[@#tbl:tbl_boot_progress_codes] and [@#tbl:tbl_boot_error_codes] describe some common combinations of

Boot Progress Codes and Boot Error Codes that can be used. For the raw values of these definitions, refer to PI Specification [49] and to Section F.2

Table 35: Boot Progress Codes

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
Setup Verifying Password	EFI_SOFTWARE_DXE_BS_DRIVER | EFI_SW_DXE_BS_PC_VERIFYING_PASSWORD

Name	Progress Code
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

Table 36: Boot 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

G Trusted Communication Between MC and Server System Devices

The information in this section is provided for guidance only. The dependencies and security relation between MCs and managed system devices are platform specific. The sequence and timing of data exchange between MCs and system devices are IMPLEMENTATION DEFINED. Communication between MCs follows the MCs to a system device communication model. In this section, an MC may refer to any Management Controller in the system, such as a BMC or a SatMC.

Server systems that operate with confidential or platform sensitive data should consider including additional security features for protecting the integrity and validity of data exchanged between MC and system devices. This section describes use cases in which MC and a system device should implement an additional security mechanism for protecting the integrity of system data. Examples of such mechanisms are attestation, device measurement, and data encryption.

MC should query a device's status to ensure that the system device boot sequence is complete. MC should initiate at least one attestation/measurement request to the system device during system runtime. For system devices that support runtime firmware update, configuration update or reset, MC should initiate attestation/measurement of the system device for every firmware update or reset event.

The use cases assume that the implementation of MC and the system device includes support for the specific security related request and data exchange protocol. Both MC and the device should provide a mechanism for querying available security features, such as SPDM over MCTP.

G.1 MC and server system device attestation

If MC should ensure the validity of a device's identity before initiating data exchange with the system device, MC should request authentication data from this device. The authentication of the system device should be verifiable using a certificate or chain of certificates and issuing a unique challenge request to this device. The attestation procedure allows MC to confirm that the target device is authentic and has not been altered or replaced.

For the attestation data exchange diagram, refer to SPDM Specification (v1.1.0, §211 and §292) [32].

G.2 MC and server system device mutual attestation

If MC and the system device should ensure the validity of their identity before initiating data exchange, mutual authentication should be initiated. The authentication of MC and the system device should be verifiable using a certificate or chain of certificates and issuing a unique challenge request to each other. The mutual attestation procedure allows MC and the system device to confirm that they are both authentic and have not been altered or replaced.

For the mutual attestation data exchange diagram, refer to the SPDM specification (v1.1.0, §306) [32].

G.3 MC and server system device measurement

If MC should ensure that the system device has a valid version of firmware(s) and configuration data, MC should send a request for measurements to this device. It is recommended that MC initiates a device attestation procedure before the device measurement request. The device measurement data may allow MC to decide to disable communication with devices with unknown, altered, or outdated firmware with possible security issue.

For the measurement data exchange diagram, refer to SPDM Specification (v1.1.0, §319) [32].

G.4 Data encryption between MC and server system device

If MC and the system device should transfer and process system sensitive data, confidential data, or system critical commands, the data traffic should be protected from being captured or altered during the transmission between MC and this device. MC and the system device should be able to negotiate encryption parameters for each session. It is recommended that MC initiates a device attestation procedure before setting up encrypted communication with this device.

For setting up a secure session, refer to SPDM Specification (v1.1.0, §95) [32].

For secure message format, refer to Secure Messages Using SPDM Specification (v1.0.0, §50) [34].

H Firmware Update

This appendix provides guidance about firmware update on SBMR compliant systems.

X The Server lifecycle management requires the firmware to be managed, which includes reporting the installed firmware inventory.

There can be several firmware images in a server. Each firmware image type is typically kept at rest in a specific non-volatile memory.

A server can contain several non-volatile memory regions where firmware images are kept. These non-volatile memories can be:

- BMC owned.
- Peripheral device owned.
- Host owned.

The firmware can be updated following a Host-based or BMC-based firmware update procedure.

I It is recommended that the Host-based and BMC-based flows do not co-exist on a server. It is otherwise challenging to keep the two flows synchronized.

H.1 Host-based firmware update

The firmware update package originates in the Host. The Host firmware is responsible for writing the firmware images either directly or indirectly using the SatMC.

The Host-based firmware update flow is described in [50].

H.2 BMC-based firmware update

The firmware update package is received by the BMC over a Redfish interface, or alternatively from the Host using the Redfish host interface, as described in [51]. Alternatively, an IMPLEMENTATION DEFINED method, such as IPMI OEM commands, can be used. If the server supports Redfish, then this is the recommended medium for firmware update package delivery to the server.

The BMC orchestrates the firmware image writes to the non-volatile memory where the image is kept at rest.

Depending on which non-volatile memory the image type is kept at rest, the BMC will either:

- Use PLDM for firmware update messaging over the BMC-IO interface to transfer the firmware images to the non-volatile memory controlled by a peripheral device [38], if the server complies with level M3 or higher. If a PLDM/MCTP communication channel does not exist, then an IMPLEMENTATION DEFINED communication protocol is used. Note that CXL and NVMe devices may use CXL [46] and NVMe [41] specific messaging over a MCTP channel for firmware update.
  - Example subsystems that can have their firmware updated in this manner: PCIe devices (such as network, storage, GPU, and NVMe) as well as CXL devices.
- Directly commit the updated firmware images to the non-volatile memory controlled by the BMC. The BMC should take care when overwriting data that could be accessed by another entity.
  - Example subsystems that can have their firmware updated in this manner: Host, SatMC FW and PSUs.

H.3 Firmware inventory

The different firmware images are directly observable by the entity that owns the non-volatile memory where the firmware images reside. The entities that own non-volatile memory, containing firmware, should provide a mechanism for the images to be discovered by relevant entities in the server.

The Host and any peripheral device are recommended to use PLDM for firmware update messaging [38], over the BMC-IO interface, to present the firmware inventory to a BMC or SatMC. If the BMC-IO interface does not support PLDM/MCTP, then an IMPLEMENTATION DEFINED mechanism may be used instead. The Host can opt to provide FW inventory to the BMC through SMBIOS [12], but that is not recommended for components that may be updated dynamically.

Note that CXL and NVMe devices may use CXL [46] and NVMe [41] specific messaging over a MCTP channel for firmware discovery.

NVMe and CXL devices can be hot-plugged. At a device hotplug event, the firmware discovery should be performed.

The BMC exposes the firmware inventory to an external entity using Redfish [51].