

Arm[®] Server Base Manageability Requirements 2.1

Platform Design Document

Non-confidential



Contents

Release information	6
Arm Non-Confidential Document License (“License”)	8
About this document	10
Terms and abbreviations	10
References	12
Cross references	14
Rules-based writing	14
Content item identifiers	14
Content item rendering	15
Content item classes	15
Progressive terminology commitment	16
Feedback	16
1 Scope and background	17
1.1 Scope	17
1.2 Background	18
1.2.1 Host SoC in-band interface	18
1.2.2 SoC side-band interface	19
1.2.3 Host-to-SatMC interface	20
1.2.4 PCIe connection between the Arm SoC and the BMC	20
1.2.5 USB connection between the Arm SoC and the BMC	20
1.2.6 JTAG connection between the Arm SoC and the BMC	20
1.2.7 Additional connectivity between the Arm SoC and the BMC	21
1.2.8 Multi-socket platform	21
1.2.9 Considerations for MCTP over SMBus/I2C and I3C	21
1.3 Arm SoC-BMC interface terminology	21
2 Compliance levels and requirements	23
2.1 Level M1	26
2.1.1 SoC-BMC interface	27
2.1.2 BMC-platform elements interface	28
2.1.3 BMC management services (out-of-band) interface	28
2.2 Level M2	29
2.2.1 SoC-BMC interfaces	30
2.2.2 BMC-platform elements interface	30
2.2.3 BMC-IO device interface	30
2.2.4 BMC management services (out-of-band) interface	30
2.3 Level M2.1	32
2.3.1 SoC-BMC interfaces	33
2.3.2 BMC-platform elements interface	33
2.3.3 BMC-IO device interface	34
2.3.4 BMC management services (out-of-band) interface	34
2.4 Level M3	35
2.4.1 SoC-BMC interface	36
2.4.2 BMC-platform elements interface	37
2.4.3 BMC-IO device interface	37
2.4.4 BMC management services (out-of-band) interface	38
2.4.5 SPDm over MCTP for BMC and side-band devices	38
2.5 Level M4	40
2.5.1 SoC-BMC interface	41
2.5.2 BMC-IO device interface	41
2.5.3 BMC-platform elements interface	41

2.5.4	BMC management services (out-of-band) interface	42
2.5.5	SPDM over MCTP for BMC and side-band devices	42
2.6	Level M5a	43
2.6.1	SoC-BMC interface	43
2.6.2	BMC-IO device interface	44
2.6.3	BMC-platform elements interface	45
2.6.4	BMC management services (out-of-band) interface	45
2.6.5	SPDM over MCTP for BMC and side-band devices	45
2.6.6	Host-to-SatMC interface	45
2.7	SBMR checklist	45
2.7.1	SBMR Level M1 checklist	46
2.7.2	SBMR Level M2 checklist	46
2.7.3	SBMR Level M2.1 checklist	46
2.7.4	SBMR Level M3 checklist	47
2.7.5	SBMR Level M4 checklist	47
2.7.6	SBMR Level M5a checklist	48
A	OpenBMC	49
B	IPMI	50
B.1	Standard IPMI commands	50
B.1.1	Remote power control	50
B.1.2	Boot device selection	50
B.1.3	BMC to host mapping	50
B.1.4	BMC user manipulation	50
B.1.5	Redfish host interface credentials bootstrapping	51
B.1.6	IPMI support verification	51
B.2	Arm standard IPMI commands	51
B.2.1	General IPMI commands format	51
B.2.2	List of Arm standard IPMI commands	52
B.3	IPMI specification clarifications and corrections	52
B.4	SSIF single and multi-part transactions	52
C	RAS	55
C.1	Level M1	55
C.1.1	SMBus System Interface (SSIF) in-band interface	55
C.1.2	RAS IPMI message format	56
C.1.3	SoC side-band interface	57
C.1.4	Out-of-band interface	57
C.2	Level M2 and Level M2.1	57
C.2.1	Redfish and IPMI host (in-band) interfaces	58
C.2.2	RAS Redfish message format	58
C.2.3	SoC side-band interface	59
C.2.4	Out-of-band interface	59
C.3	Level M3, M4, and M5a	59
C.3.1	Redfish host (in-band) interface	60
C.3.2	MCTP and PLDM (SoC side-band) interface	60
C.3.3	Out-of-band interface	67
D	Platform monitoring and control	69
D.1	Background	69
D.2	IPMI commands to monitor and control managed entities	69
D.3	Redfish schema to monitor and control managed entities	70
D.4	PLDM commands/APIs to monitor and control managed entities	70
D.4.1	Examples of PLDM sensors exposed by SatMC	74

E	Reference implementation of remote debug using OpenOCD	75
E.1	Background	75
E.2	Levels M1, M2, M2.1, M3, M4, M5a	75
F	Boot progress codes	77
F.1	IPMI commands for boot progress codes	77
F.1.1	Send boot progress code (NetFn 2Ch, Command 02h)	77
F.1.2	Get boot progress code (NetFn 2Ch, Command 03h)	77
F.2	Boot progress code format	78
F.2.1	IPMI progress code definition	79
F.2.2	PLDM progress code definition	79
F.2.3	Example progress codes (IPMI, PLDM)	80
F.2.4	Example boot progress codes (Redfish)	82
F.3	Common boot progress codes	82
G	Trusted communication between MC and system devices	85
G.1	MC and server system device attestation	85
G.2	MC and server system device mutual attestation	85
G.3	MC and server system device measurement	85
G.4	Data encryption between MC and server system device	86
G.5	PA-RoT and AC-RoT responsibilities	86
G.5.1	PA-RoT responsibilities	86
G.5.2	AC-RoT responsibilities	86
H	Firmware update	87
H.1	Host-based firmware update	87
H.2	BMC-based firmware update	87
H.3	Firmware inventory	87

Copyright © 2020-2024 Arm Limited. All rights reserved.

Release information

Version Issue E, 2.1 (10 Oct 2024)

- Introduce the concepts of PA-RoT and AC-RoT to Level M5 (461).
- Add MMBI and MCTP over MMBI as an option for host in-band interface for Level M5 (405).
- Add MCTP over PCC as an option between host and SatMC for Level M5 (651).
- Add MCTP over USB as an option for side-band interface in Level M5 (456).
- Add MCTP over USB as an option for BMC to Platform Elements communication in Level M5 (714).
- Add MCTP over USB as an option for BMC-IO communication in Level M5 (683).
- Add optional UART between SatMC and BMC in Level M5 (457).
- Define boot progress code transfer over PLDM (370).
- Require BIOS Redfish settings for Level M5 (354).
- Recommend BMC to be the I3C Side-band primary controller (467).
- Deprecate SBMR Level M0 (746).
- Relax SSIF SMBAlert requirement to a recommendation (652).
- Relax requirement on boot progress code over IPMI (721).
- Relax IPMI boot targets requirements (653).
- Add an M3 note that side-band interface is bi-directional (692).
- Update OCP Baseline Profile requirements to latest version (654).
- Recommendation of minimizing MUX usage for MCTP over I2C/I3C to improve performance for usability (715).
- Use PLDM CPEREvent standard event class definition (413).
- Update CPER Redfish schema requirements (734).
- Errata for M21_IPMI1, IPMI_8, and checklist inclusion of M1_RAS_1, M1_RAS_2, M2_RAS_1, M2_RAS2, and M3_RAS_1 (600).
- Errata for NC-SI over RBT and MCTP across SBMR Levels (773).
- Remove deprecated terms for Arm inclusive language commitments (713).
- Clarify that IPMI SSIF is over SMBus, not just I2C (792).

Version Issue D, 2.0 (27 Apr 2022)

- 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 paper reference (403).
- Remove SBSG reference (400).
- Inclusive language considerations (401).

Version Issue C, 1.1 (11 Feb 2021)

- 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

Version Issue B, 1.0 (15 Jun 2020)

- License LES-PRE-21585

Version Issue A, 1.0 (30 Jan 2020)

- Initial release, SBMR 1.0

Arm Non-Confidential Document License (“License”)

This License 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 License (“**Document**”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

“**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 License between you and Arm.

Subject to the terms and conditions of this License, 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 License 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 License granted in (i) above; and
- (iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses 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 content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm’s view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

Reference by Arm to any third party’s products or services within this document is not an express or implied approval or endorsement of the use thereof.

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

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (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 LICENSE). 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 License 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 License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License 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 License, all terms shall survive except for the License grants.

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

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

This License may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this License and any translation, the terms of the English version of this License shall prevail.

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

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

Copyright © 2020-2024 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

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] Management Component Transport Protocol (MCTP) [4], and Security Protocol and Data Model (SPDM) [5]. These specifications are defined in the DMTF Redfish Forum, Platform Management Components Intercommunication (PMCI), and Security Protocols and Data Modles (SPDM) Working Groups.

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AC-RoT	Active Component Root of Trust
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	Compute Express Link 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.
MMBI	Memory-Mapped BMC Interface. An interface for communication between a Management Controller (typically a BMC)

Term	Meaning
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. See [6].
Node	For the purpose of this specification, a node is a single server system in a group of managed servers.
OCP	Open Compute Project
OEM	Original Equipment Manufacturer. In this document, the final device manufacturer.
PA-RoT	Platform Active Root of Trust
PCC	ACPI Platform Communications Channel. See [7].
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.
PMCI	Platform Management Component Intercommunication. A working group within the DMTF industry standards forum, that defines standards to address “inside the box” communication interfaces between the components of the platform management subsystem.
RBT	RMII-Based Transport. Electrical and timing specification for a physical medium that is derived from the Reduced Media Independent Interface (RMII) transport.
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.
Requester	An agent in a computing system that is capable of initiating memory transactions.
RMII	Reduced Media Independent Interface (RMII)
SatMC	Satellite Management Controller. A microcontroller or processor that interpret and process management-related data, and initiate management-related actions on management devices. It can be part of SoC or can be outside of SoC.
SBSA	Server Base System Architecture.
SiP	Silicon Partner. In this document, the silicon manufacturer.
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. See [8]

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 Platform Level Data Model (PLDM) Base Specification*. DMTF.
- [4] *DSP0236 Management Component Transport Protocol (MCTP) Base Specification*. DMTF.
- [5] *DSP0274 Security Protocol and Data Model (SPDM) Specification*. DMTF.
- [6] *DSP0222 Network Controller Sideband Interface (NC-SI) Specification*. DMTF.
- [7] *Advanced Configuration and Power Interface (ACPI) Specification*. UEFI Forum.
- [8] *Unified Extensible Firmware Interface (UEFI) Specification*. UEFI Forum.
- [9] *DSP8010 Redfish Schema Bundle*. DMTF.
- [10] *Intelligent Platform Management Interface (IPMI) 2.0, Revision 1.1 (October 2013)*. Dell, HP, Intel, NEC.
- [11] *OCF Baseline Hardware Management Redfish Profile*. Open Compute Project.
- [12] *OCF Server Hardware Management Redfish Profile*. Open Compute Project.
- [13] *DSP2015 Platform Management Communications Infrastructure (PMCI) Architecture White Paper*. DMTF.
- [14] *DSP0134 SMBIOS Reference Specification*. DMTF.
- [15] *DSP0256 Management Component Transport Protocol (MCTP) Host Interface Specification*. DMTF.
- [16] *DSP0282 Memory-Mapped BMC Interface (MMBI) Specification*. DMTF.
- [17] *DSP0284 Management Component Transport Protocol (MCTP) Memory-Mapped BMC Interface (MMBI) Transport Binding Specification*. DMTF.
- [18] *DSP0238 Management Component Transport Protocol (MCTP) PCIe VDM Transport Binding Specification*. DMTF.
- [19] *DEN 0101 Authenticated Debug Access Control Specification (ADAC)*. Arm Ltd.
- [20] *DEN 0094 Arm Base System Architecture (BSA)*. Arm Ltd.
- [21] *OCF Server Designs and Specifications*. Open Compute Project.
- [22] *OCF Datacenter Secure Control Module Specification*. Open Compute Project.
- [23] *System Management Bus (SMBus) Specification*. System Management Interface Forum.
- [24] *DEN 0044 Arm Base Boot Requirements (BBR)*. Arm Ltd.
- [25] *Arm IHI 0031 Arm Debug Interface Architecture Specification, ADiv5*. Arm Ltd.

- [26] *Arm IHI 0074 Arm Debug Interface Architecture Specification, ADIv6*. Arm Ltd.
- [27] *DSP0270 Redfish Host Interface Specification*. DMTF.
- [28] *DSP0272 Redfish Interoperability Profile Specification*. DMTF.
- [29] *DSP8013 Redfish Interoperability Profiles Bundle*. DMTF.
- [30] *DSP2046 Redfish Resource and Schema Guide*. DMTF.
- [31] *DSP0245 Platform Level Data Model (PLDM) IDs and Codes Specification*. DMTF.
- [32] *DSP0248 Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification*. DMTF.
- [33] *DSP0249 Platform Level Data Model (PLDM) State Set Specification*. DMTF.
- [34] *DSP0239 Management Component Transport Protocol (MCTP) IDs and Codes*. DMTF.
- [35] *DSP0241 Platform Level Data Model (PLDM) Over MCTP Binding Specification*. DMTF.
- [36] *DSP0275 Security Protocol and Data Model (SPDM) over MCTP Binding Specification*. DMTF.
- [37] *DSP0277 Secured Messages using SPDM Specification*. DMTF.
- [38] *DSP0276 Secured Messages using SPDM over MCTP Binding Specification*. DMTF.
- [39] *DSP0237 Management Component Transport Protocol (MCTP) SMBus/I2C Transport Binding Specification*. DMTF.
- [40] *DSP0233 Management Component Transport Protocol (MCTP) I3C Transport Binding Specification*. DMTF.
- [41] *DSP0267 Platform Level Data Model (PLDM) for Firmware Update Specification*. DMTF.
- [42] *DSP0218 Platform Level Data Model (PLDM) for Redfish Device Enablement Specification*. DMTF.
- [43] *NVM Express Management Interface*. NVM Express.
- [44] *DSP0235 NVMe Management Messages over MCTP Binding Specification*. DMTF.
- [45] *OCF Usage Guide for Server Profile*. Open Compute Project.
- [46] *OCF Usage Guide for Baseline Profile*. Open Compute Project.
- [47] *CXL Specification*. Compute Express Link.
- [48] *CXL Type 3 Management Using MCTP CCI ECN*. Compute Express Link.
- [49] *DSP0234 CXL Fabric Manager API over MCTP Binding Specification*. DMTF.
- [50] *DSP0281 CXL Type 3 Device Component Command Interface over MCTP Binding Specification*. DMTF.
- [51] *DSP0283 Management Component Transport Protocol (MCTP) Universal Serial Bus (USB) Transport Binding Specification*. DMTF.
- [52] *Architectural Out-of-Band Management ECN*. PCI-SIG.

- [53] [OCP Secure Firmware Recovery specification](#). Open Compute Project.
- [54] [DSP0292 Management Component Transport Protocol \(MCTP\) PCC Transport Binding Specification](#). DMTF.
- [55] [DSP0268 Redfish Data Model Specification](#). DMTF.
- [56] [Open On-Chip Debugger \(OpenOCD\) Project](#). OpenOCD Project.
- [57] [Platform Initialization \(PI\) Specification](#). UEFI Forum.
- [58] [DEN 0118 Platform Security Firmware Update for the A-profile Arm Architecture](#). Arm Ltd.
- [59] [DSP2062 Redfish Firmware Update White Paper](#). 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 [7] section 5.6.6
- UEFI § 6.1 - Reference to the UEFI specification [8] section 6.1

Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Declaration
- Rule
- Goal
- Information
- Rationale
- Implementation note
- Software usage

Declarations and Rules are normative statements. An implementation that is compliant with this specification must conform to all Declarations and Rules in this specification that apply to that implementation.

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and Rules, these are generally grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

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

Content items other than Declarations and Rules are informative statements. These are provided as an aid to understanding this specification.

Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: L_{iiii}

- L is a label that indicates the content class of the content item.
- $iiii$ is the identifier of the content item.

Content item classes

Declaration

A Declaration is a statement that does one or more of the following:

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behaviour.

A Declaration is rendered with the label D .

Rule

A Rule is a statement that describes the behaviour of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label R .

Goal

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label G .

Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label I .

Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label X .

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label U .

Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label S .

Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

Previous issues of this document included terms that can be offensive. We have replaced these terms. If you find offensive terms in this document, please contact terms@arm.com.

Feedback

Arm welcomes feedback on its documentation.

If you have any comments or suggestions for additions and improvements create a ticket at

<https://support.developer.arm.com>.

As part of the ticket include:

- The title (Server Base Manageability Requirements).
- The document ID and version (DEN0069E 2.1).
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Scope and background

This document provides a path to establish a common foundation for server management on SBSA-compliant Arm AArch64 servers where common capabilities are standardized and differentiation truly valuable to the end-users 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 [9] defined using JSON Schema, OData 4.0, or OpenAPI formats. Redfish provides a secure, multi-node capable replacement for IPMI-over-LAN [10]. It is intended to meet Open Compute Project (OCP) [11] [12] 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, see [13]. Figure 3 in that document 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) [10] 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 might make the IPMI support optional.

This document addresses the need to establish the following common standard interface sets. Figure 1 shows a high-level diagram of these interface sets:

1. Arm SoC-BMC 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, such as devices and 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 system admins via external network to manage servers remotely.

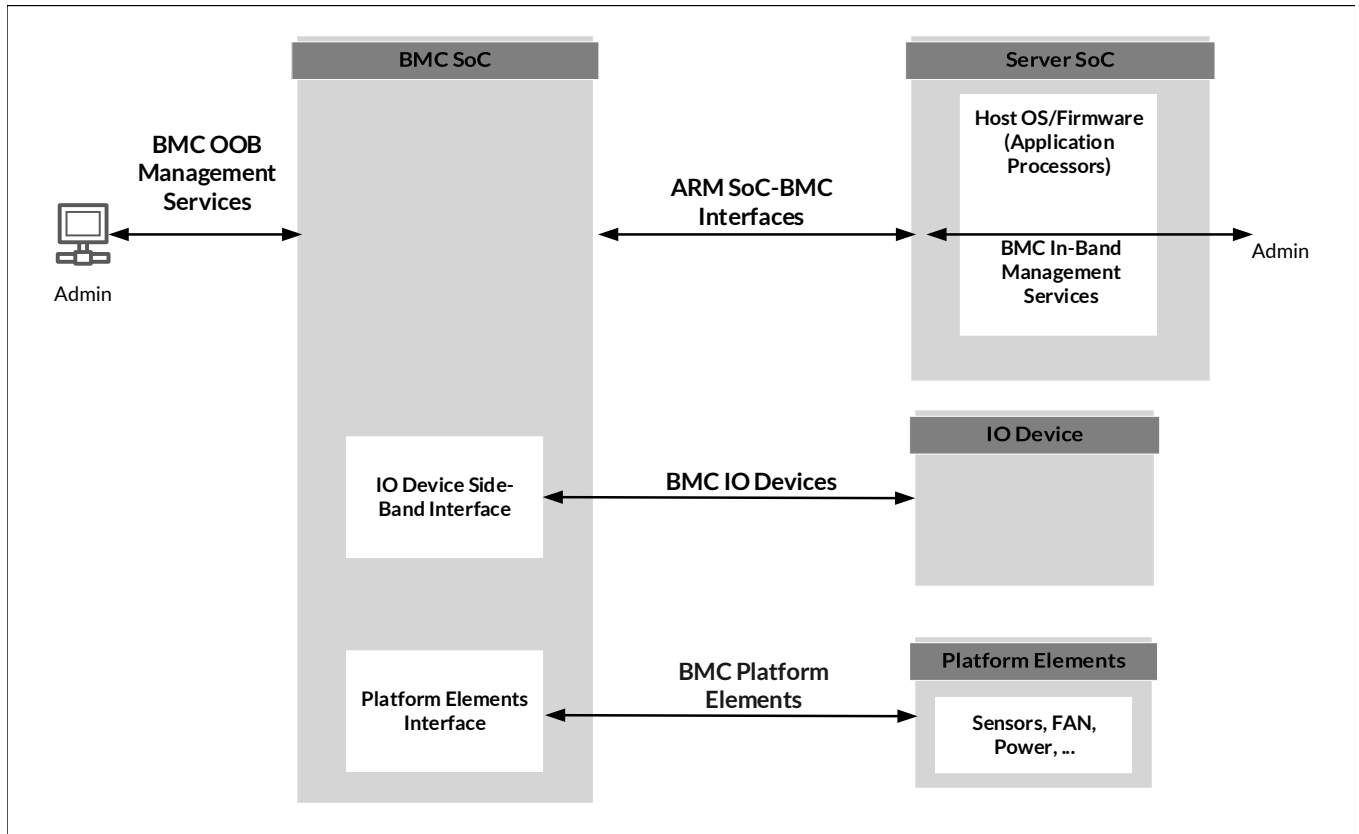


Figure 1: Server Management Interfaces

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, and User Software, as well as System Firmware, such as UEFI [8], to communicate with the BMC. It is typically exposed to Host Software via SMBIOS [14], ACPI [7] tables (such as SPMI), and/or PCIe configuration space. Arm server systems typically use one or more of the following in-band interfaces:

- IPMI SSIF Host Interface [10]
- Redfish Host Interface [2]
- MCTP Host Interface [15] over an in-band physical medium, such as Memory-Mapped BMC Interface (MMBI) [16] [17].

The typical use-cases for this interface are described in Table 2 below.

Table 2: Use cases of host SoC in-band interfaces

Communication entities	IPMI SSIF	MCTP Host Interface	Redfish Host Interface	Usage examples
UEFI - BMC communication	Yes	Yes	Yes (not standard)	Reporting SMBIOS [14] table
	Yes	Yes		Reporting boot progress codes
	Yes	Yes		Error reporting (in some cases)
	Yes	Yes		General event logging
		Yes	Yes	BIOS Configuration settings
	Yes	Yes	Yes (not standard)	General UEFI - BMC data exchange
OS/Hypervisor - BMC communication	Yes			Reading IPMI event log
	Yes			Redfish Authentication
	Yes	Yes		Error reporting (in some cases)
	Yes	Yes		Some other event logging
User software - BMC communication	Yes		Yes	Local server configuration, update, deployment, monitoring, without going through the OOB interface.

U

Sending large amount of data (such as SMBIOS table) over a slow interface, such as the IPMI SSIF I2C bus, may impact the boot time. Implementations may choose alternative interfaces for these use-cases, including:

- MCTP Host Interface over a higher bandwidth host interface, such as MMBI [16] [17].
- 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 - BMC communication using non-standard OEM Redfish schema extensions.
- Other IMPLEMENTATION DEFINED interfaces, such as PCIe-based mailbox or shared memory.

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 include:

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

U

It is also possible for the side-band 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.

U

This interface can vary in bandwidth capabilities depending on the physical interface used. For example, USB 2.0 provides up to 480 Mbps communication, which is faster than I3C (12.5 Mbps) and I2C/SMBUs (in the order of 100s of Kbps).

1.2.3 Host-to-SatMC interface

The Host-to-SatMC interface is part of the SoC side-band interface. Its primary purpose is to allow the Host and SatMC to exchange information. The typical use-cases are:

- RAS error reporting (for OS-first RAS errors).
- Platform attestation (where the PProT is accessible via the SatMC).

1.2.4 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 [18]. In this usage, the Arm SoC must contain the logic to route the PCIe VDM messages to the proper IO devices.
- Shared standard memory mailbox communication between the BMC and the SoC host software, using MMBI [16] and MCTP over MMBI [17].
- 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.

1.2.5 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

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.

1.2.6 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) [19].

1.2.7 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

1.2.8 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.

1.2.9 Considerations for MCTP over SMBus/I2C and I3C

MCTP [4] is used as a transport protocol format that is independent of the underlying physical bus and data-link layer messaging used on the bus. It is used for intercommunication between various elements of the management subsystem. The physical bus MCTP communication can vary, and include SMBus/I2C, I3C, PCIe VDM, and USB, among others.

To improve performance and usability, it is recommended to minimize the use of multiplexers for MCTP over SMBus/I2C and I3C. With I2C and I3C multiplexers, accesses to MCTP devices are limited to one device at a time, as the MCTP transactions are blocking until completion.

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 can leverage a proprietary protocol or a more standard transport protocol, such as MCTP/PLDM. This is a multi-requester 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

Name	Requester	Completer	Description / Example / Notes	In SBMR Scope?
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 (for example: 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 can be bi-directional (for example, PROCHOT).	Partially Covered
SoC notification pins (for example: 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 can be bi-directional (for example, 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 [20] 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	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 can 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	BMC	Platform Elements	This can 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, M4, and M5a. Unless otherwise stated in this specification, each level builds upon the requirements of the previous (lower) level, with any additional requirements or exceptions documented in each level.

Note

M5a describes preliminary definitions of a future compliance level, for the purpose of public review and feedback. The 'a' denotes that this is an alpha work in-progress compliance level. These definitions are subject to change in future publications of this specification.

Table 4 below shows the summary of SBMR Compliance levels.

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, see 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	Host-to-SatMC Interface
M1	Required IPMI	IMPLEMENTATION DEFINED	Required: IPMI SSIF.	IMPLEMENTATION DEFINED	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 over RBT is required.	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED
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) is required.	IMPLEMENTATION DEFINED see [21], [22] and [10] for guidance.	IMPLEMENTATION DEFINED

	Out-of-band Level Interface	SoC Side-band Interface	Host/SoC In-band Interface	BMC IO Device Interface	BMC Platform Element Interface	Host-to-SatMC 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		
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 RBT or MCTP (over I3C or PCIe VDM) is required. 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.	IMPLEMENTATION DEFINED IMPLEMENTATION DEFINED See [21], [22] and [10] for guidance. 	

	Out-of-band Level Interface	SoC Side-band Interface	Host/SoC In-band Interface	BMC IO Device Interface	BMC Platform Element Interface	Host-to-SatMC Interface
				Other IO Devices IMPLE- MENTATION DEFINED.		
M5a	Required: Redfish.	Required: MCTP/PLDM over USB, I3C, or PCIe VDM. SMBus/I2C can be used only for initial discovery or as a fallback.	Required: IPMI SSIF and Redfish Host Interface. Recommended: MCTP over MMBI	Conditional Requirement: If shared physical NIC is used, NC-SI over RBT or MCTP (over I3C or USB or PCIe VDM) is required. Conditional Requirement: MCTP/PLDM for PCIe devices (Network and Storage), and NVMe-MI over MCTP (for NVMe disks), using I3C, USB or PCIe VDM, with I2C as fallback. Recommended: CXL FM and CCI over MCTP for CXL devices, using I3C, USB or PCIe VDM, with I2C as fallback. Other IO Devices IMPLE- MENTATION DEFINED.	IMPLEMENTATION DEFINED See [21], [22] and [10] for guidance. Recommended: PLDM/MCTP.	Recommended: MCTP/PLDM over PCC.

2.1 Level M1

I This section defines the requirements for Level M1-based servers. Figure 2 shows a summary of these requirements.

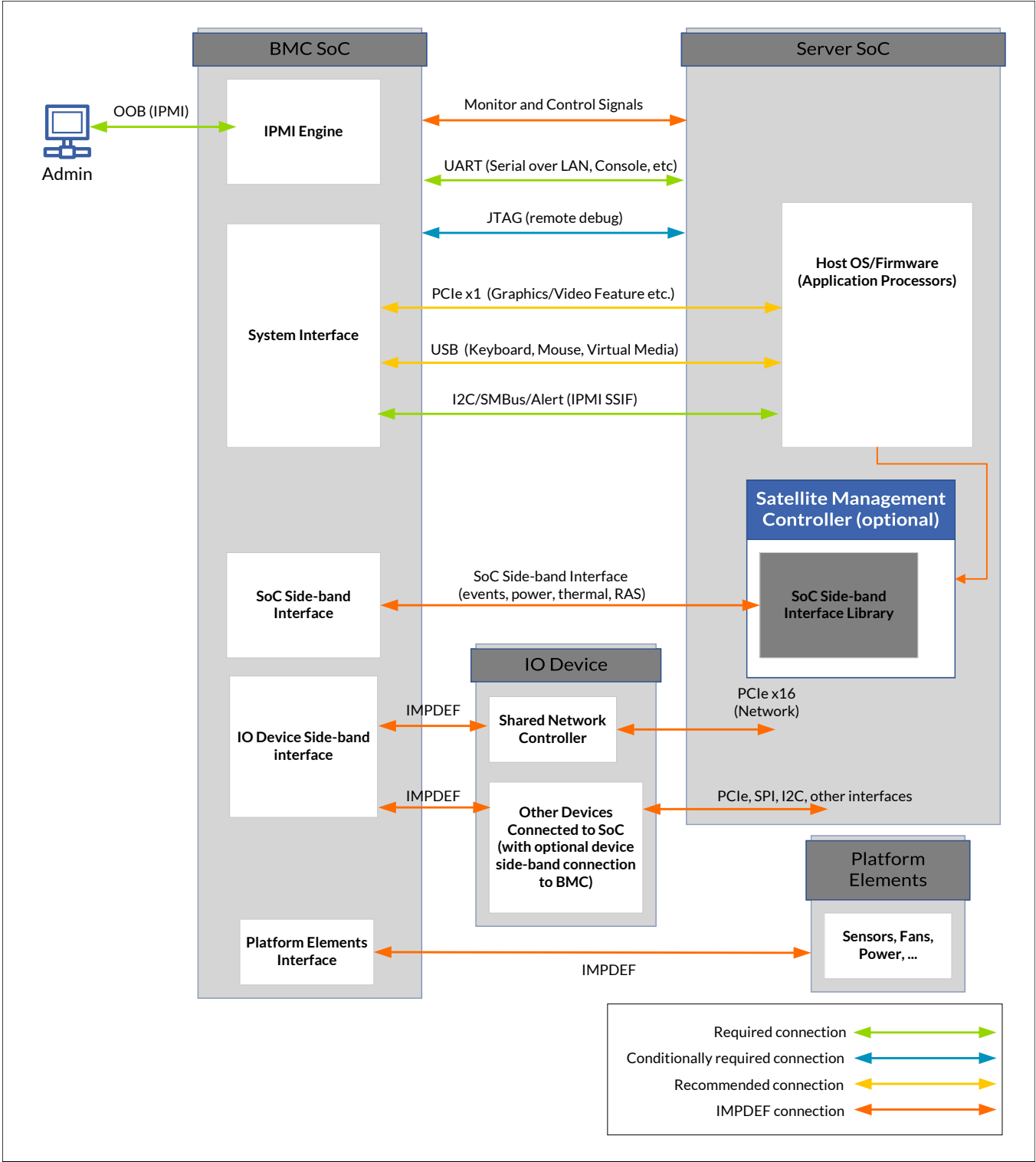


Figure 2: Server management interfaces for Level M1

2.1.1 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.

2.1.1.1 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 [10] 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 When using I2C connection for IPMI SSIF, I2C controller on Arm SoC and BMC must conform to the SMBus specification [23].

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

I The recommended SMBus completer 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.1.

2.1.1.2 Console UART

R_{M1_UART_1} The Arm SoC must have at least one BSA [20] 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 [20] and BBR [24], the console UART must be a BSA [20] compliant UART that is exposed to the host software using the Serial Port Console Redirection (SPCR) ACPI [7] 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.

2.1.1.3 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.

2.1.1.4 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.

2.1.1.5 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 [25] or ADiv6 [26].

- 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 (for example, 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, see Section E.

2.1.2 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.

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

R_{M1_OOB_1} Support for IPMI is a requirement for M1-compliant server systems.

R See Section B.1 for minimal IPMI commands required.

2.2 Level M2

I This section defines the requirements for Level M2-based servers. Figure 3 shows a summary of these requirements.

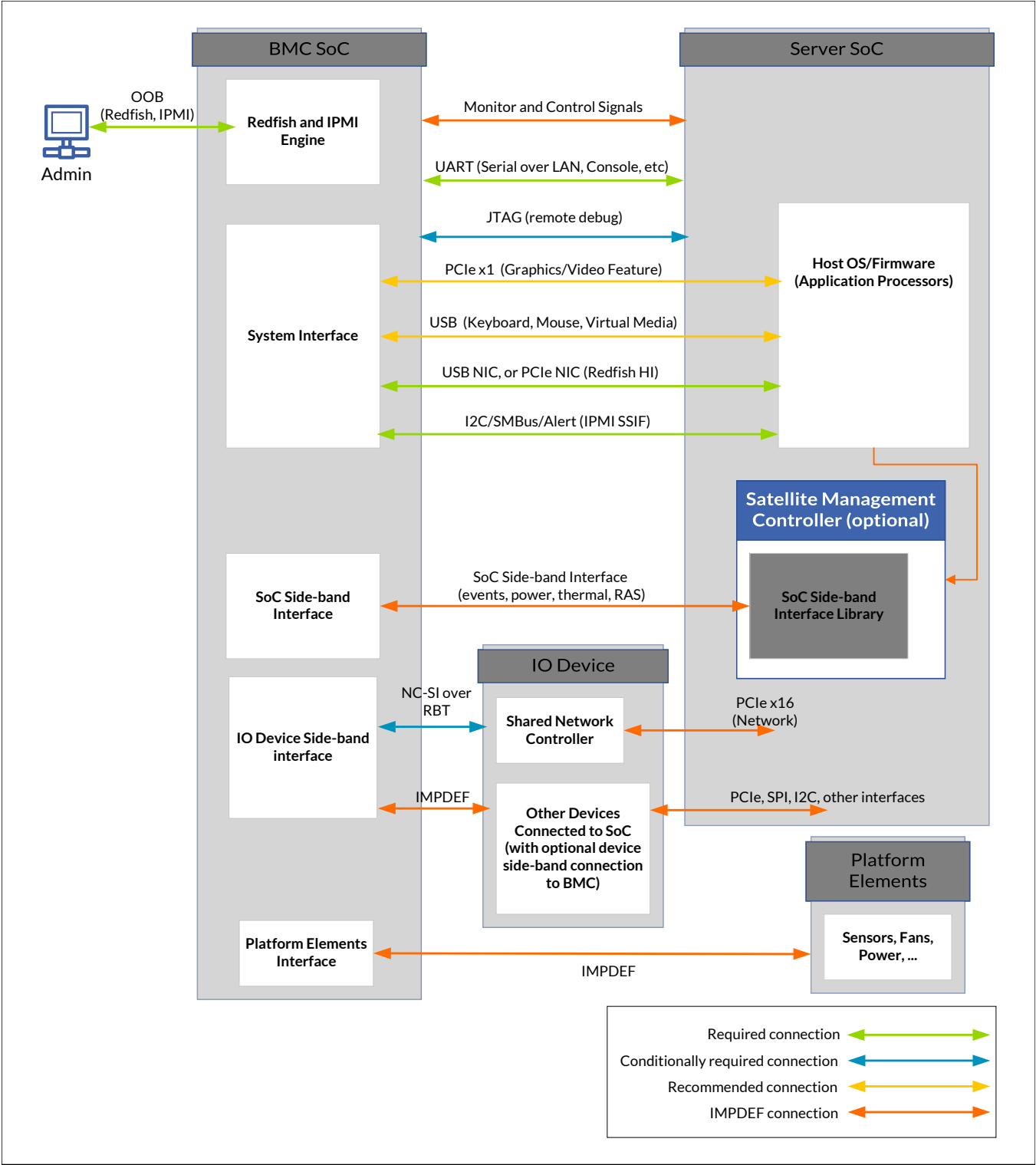


Figure 3: Server management interfaces for Level M2

2.2.1 SoC-BMC interfaces

R For Level M2-based server systems, the requirements and recommendations extend those of the level M1.

2.2.1.1 Host SoC in-band interface

R_{M2_IB_1} The Host/SoC In-Band interface must be compliant to the Redfish Host Interface Specification [27]. 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, [27] 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.

2.2.1.2 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} 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.

2.2.2 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.

2.2.3 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) [6] over Reduced Media Independent Interface (RMII) Based Transport (RBT) is required for Level M2-based server systems.

X NC-SI [6] 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.

2.2.4 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 See Section B.1 for the minimal IPMI commands required.

R _{M2_00B_3}	<p>Level M2-based server systems further standardize the BMC management services interface by adopting the Redfish Interoperability Profiles Specification [28] and the individual profiles contained in the Redfish Interoperability Profiles Bundle [29]. Supporting OpenCompute Project (OCP) defined profiles is required for OCP compliant servers. OCP currently defines two Redfish profiles for hardware management:</p> <ol style="list-style-type: none"> 1. OCP Baseline Hardware Management Redfish Profile [11] . This is the minimum level a Redfish interface must provide for OCP compliant hardware management. 2. OCP Server Hardware Management Redfish Profile [12]. This profile defines additional requirements on top of the OCP Baseline profile [11] for OCP compliant server hardware management.
X	<p>As Redfish Schema [9] 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.</p>
I	<p>Redfish Resource and Schema Guide [30] provides information on how to use the Redfish API, targeted at consumption of the API.</p>
S	<p>A DMTF tool to verify the compliance of a Redfish implementation to the required Redfish profile is available here. This is also integrated as part of the Arm SBMR Architecture Compliance Test Suite (SBMR-ACS) which is available here.</p>

Note

Arm can publish Arm-specific profiles if needed. Here, the intent is to adopt the standard profiles, such as the OCP profile [11] [12].

2.3 Level M2.1

I This section defines the requirements for Level M2.1-based servers. Figure 4 shows a summary of these requirements.

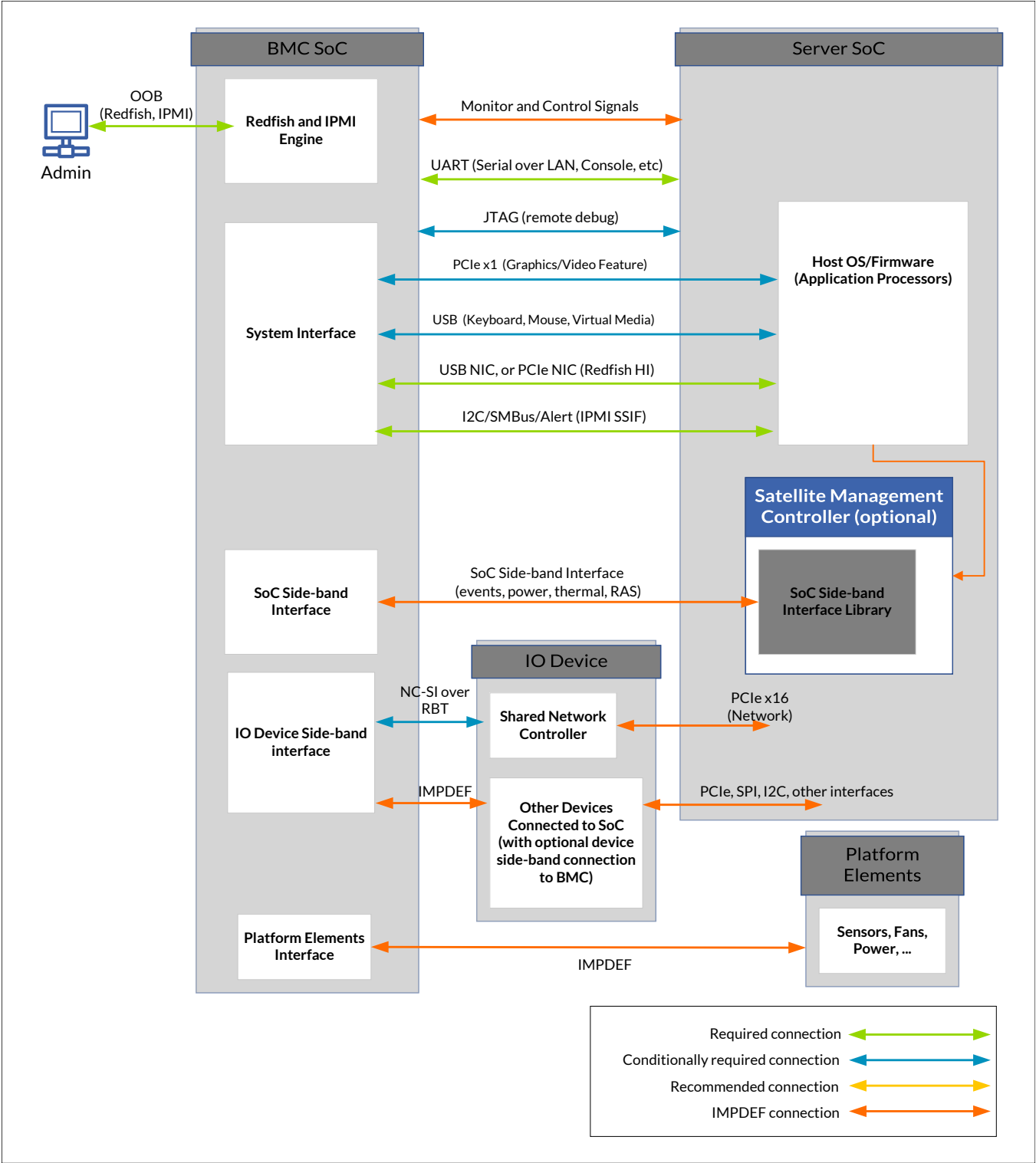


Figure 4: Server management interfaces for Level M2.1

2.3.1 SoC-BMC interfaces

R For Level M2.1-based server systems, the requirements and recommendations extend those of the level M2.

2.3.1.1 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.

I It is recommended that the SSIF interface supports 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 completer 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 for both the in-band and out-of-band interfaces:

- 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 IPMI Command (as defined in Section C.1.2). This is required only if the platform supports reporting platform errors to the BMC over the in-band interface using IPMI SSIF.
- Send Boot Progress Code IPMI Command (as defined in Section F.1). This is required only if the platform supports reporting boot progress codes to the BMC over the in-band interface using IPMI.
- Reporting Platform Error Records and Boot Progress codes using IPMI OEM commands is not permitted.

R If the platform supports reporting platform errors to the BMC using the in-band interface, then the additional rules in Section C.1 and Section C.2 must be implemented.

I A base server system may implement additional IMPLEMENTATION DEFINED methods and protocols for reporting these events. When doing so, it is recommended that the standard data format of these events is used over the IMPLEMENTATION DEFINED transport. This includes:

- Using the CPER format for platform error records, as defined in [8] (UEFI § N).
- Using the Boot Progress Code format as defined in Section F.2.

X Using standard data formats helps reduce fragmentation in BMC firmware and software that interprets the data.

2.3.1.2 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.

2.3.1.3 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.

2.3.2 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.

2.3.3 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.

2.3.4 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.

2.4 Level M3

I This section defines the requirements for Level M3-based servers. Figure 5 shows a summary of these requirements.

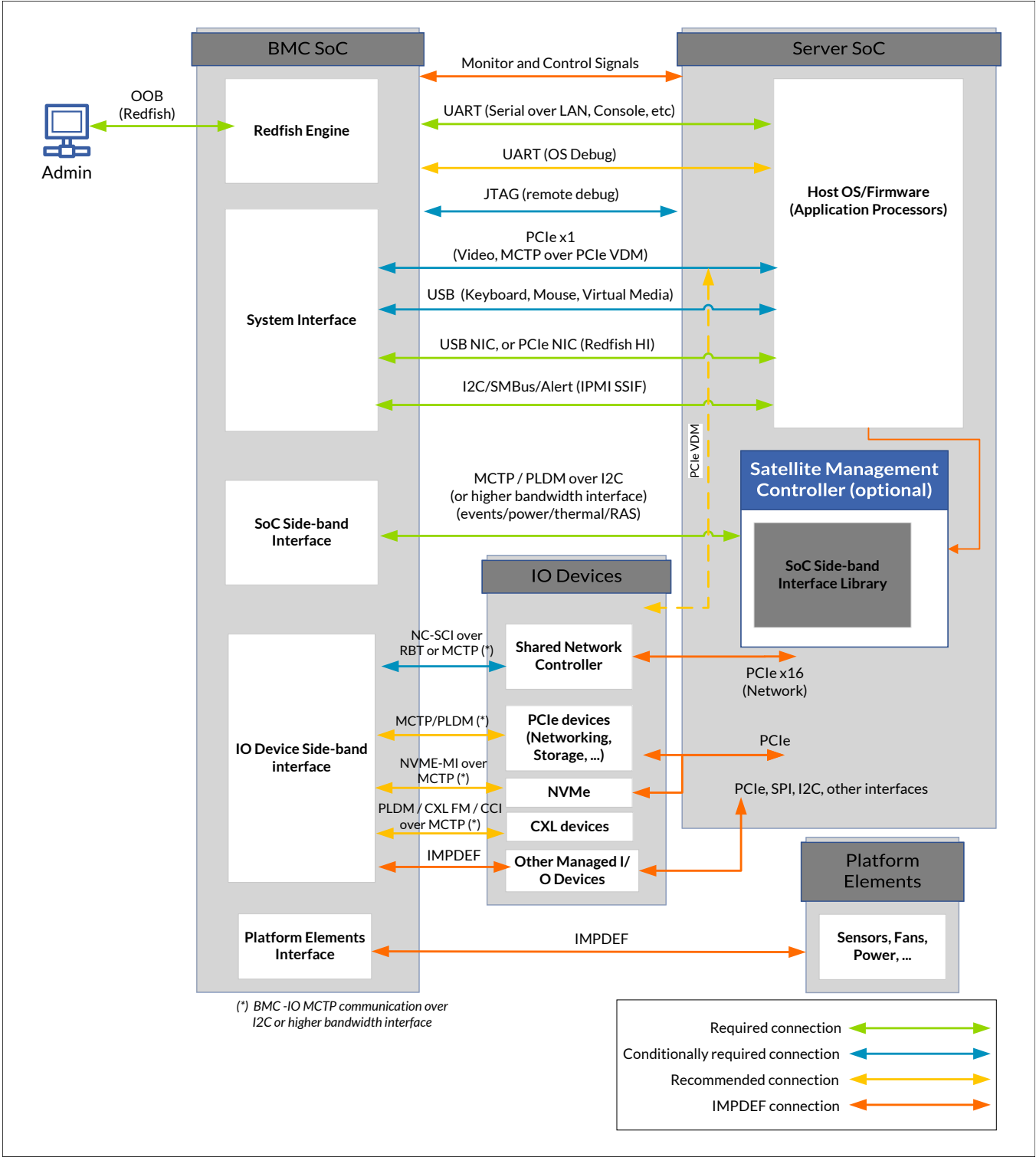


Figure 5: Server management interfaces for Level M3

2.4.1 SoC-BMC interface

R For Level M3-based server systems, the requirements and recommendations extend those of the level M2.1.

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) [22].

2.4.1.1 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.

2.4.1.2 Debug UART

I For Level M3 based systems, it is recommended that the Arm SoC has an additional BSA [20] compliant UART connection to the BMC for the purpose of remote OS debugging through the BMC.

I Per the BSA [20] and BBR [24], the debug UART must be a BSA [20] compliant UART that must be exposed to the host software using the Debug Port Table (DBG2) ACPI [7] Table. The default baud rate for interoperability with commercially available BMCs should be 115200 bits/second.

2.4.1.3 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 BMC and SatMC.

R_{M3_SB_2} PLDM [3] [31] [32] [33] 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] [34] 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 [35] is used as the format of PLDM over MCTP messages.

R_{M3_SB_5} SPDm [5] 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 [36] is used as the format of SPDm over MCTP messages.

R_{M3_SB_7} Secure messages using SPDm specifications [37] is used for the purpose of supporting secure transfer of application data over PMCI transports using SPDm. Secure messages [37] 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 [38] 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 [39]. This interface must support bi-directional transfer of MCTP packets, which requires that both sides of the communication have complete addresses. Implementations may choose a higher bandwidth physical data-link, such as MCTP over PCIe VDM [18] or MCTP over I3C [40]

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.3.

2.4.1.4 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 1.2.6.

2.4.2 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. See:

- Intelligent Platform Management Interface v2.0 (IPMI) specification [10].
- OCP server design and specifications [21].
- OCP Datacenter Secure Control Module (DC-SCM) [22].

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

2.4.3 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) [6] over RBT 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] [31] [32] [33] 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] [34] 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 [35] is used as the format of PLDM over MCTP messages.

I PLDM for Firmware Update [41] 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 [42] is used as the messages and data structures used for enabling PLDM devices to participate in Redfish-based management.

- I 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) [43] 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 [44] 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 [39]. Implementations may choose a higher bandwidth physical data-link, such as MCTP over PCIe VDM [18] or MCTP over I3C [40]
- 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.

2.4.4 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](#). This is also integrated as part of the [Arm SBMR Architecture Compliance Test Suite \(SBMR-ACS\)](#).
 - Must conform to the OCP Baseline Hardware Management Redfish Profile v1.0.1 or newer [11]. It is also recommended to conform to the OCP Server Hardware Management Interface Redfish Profile v1.0.0 or newer.
 - 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>.
 - This is also integrated as part of the [Arm SBMR Architecture Compliance Test Suite \(SBMR-ACS\)](#).
- S For more information on using the OCP profiles, see [45] and [46].

2.4.5 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 [5] version 1.1 or newer.
- R_{M3_SPDM_2} Must conform to the SPDm over MCTP binding specification [36] 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 [38]
- I BMC and side-band device may encrypt data using secure messages with SPDm over MCTP (MCTP Type 6 messages) [38] in case the SPDm secure session is established between BMC and side-band device.

2.5 Level M4

I This section defines the requirements for Level M4-based servers. Figure 6 shows a summary of these requirements.

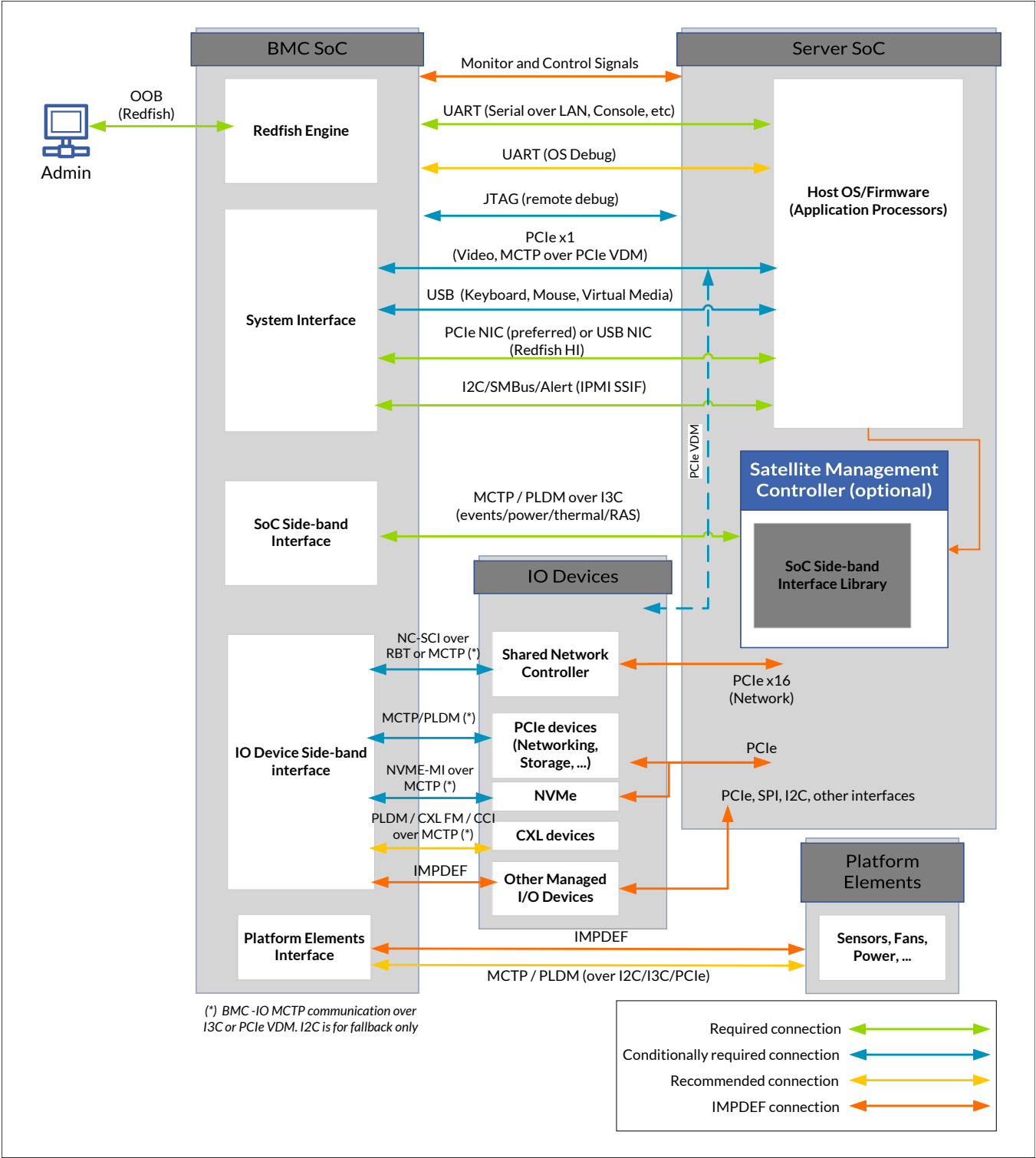


Figure 6: Server management interfaces for level M4

2.5.1 SoC-BMC interface

- R For Level M4-based server systems, the requirements and recommendations extend those of the level M3.
- 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) [22]

2.5.1.1 Host SoC in-band interface

- 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.

2.5.1.2 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 [40].
- I It is recommended that the BMC acts as an I3C Primary Controller and SatMC acts as an I3C Target. BMC as MCTP Bus Owner is responsible to discover all the I3C targets and fulfil the MCTP Bus Owner role for I3C bus.

2.5.2 BMC-IO device interface

- R The requirements and recommendations for these interfaces on Level M4 based server systems are the same as Level M3, 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 [47] [48].
- X The CXL FM API over MCTP Specification [49] 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 [47] [48] is a register interface and command set for managing CXL Type 3 devices. CXL Type 3 CCI Messages over MCTP Specification [50] 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 [40] or over PCIe VDM binding specification [18]. MCTP over SMBus/I2C [39] 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.

2.5.3 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. See:
- Intelligent Platform Management Interface v2.0 (IPMI) specification [10].
 - OCP server design and specifications [21].

- OCP Datacenter Secure Control Module (DC-SCM) [22].

- I 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, see [13].
- I PLDM [3] [31] [32] [33] 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] [34] 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 [35] is used as the format of PLDM over MCTP messages.
- I SPDm [5] 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 [36] is used as the format of SPDm over MCTP messages.
- I Secure messages using SPDm specifications [37] is used for the purpose of supporting secure transfer of application data over PMCI transports using SPDm. Secure messages [37] also defines transport requirements for SPDm records, which form the basis of encryption and message authentication.
- I Secured Messages using SPDm over MCTP binding [38] 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 [39]. Additional MCTP physical and data-link layers are defined for I3C [40] and PCIe VDM [18].
- U For a list of PLDM commands which aid in monitoring and control of platform elements, see Section D.

2.5.4 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.

2.5.5 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.

2.6 Level M5a

I This section defines the requirements for Level M5a-based servers. Figure 7 shows a summary of these requirements.

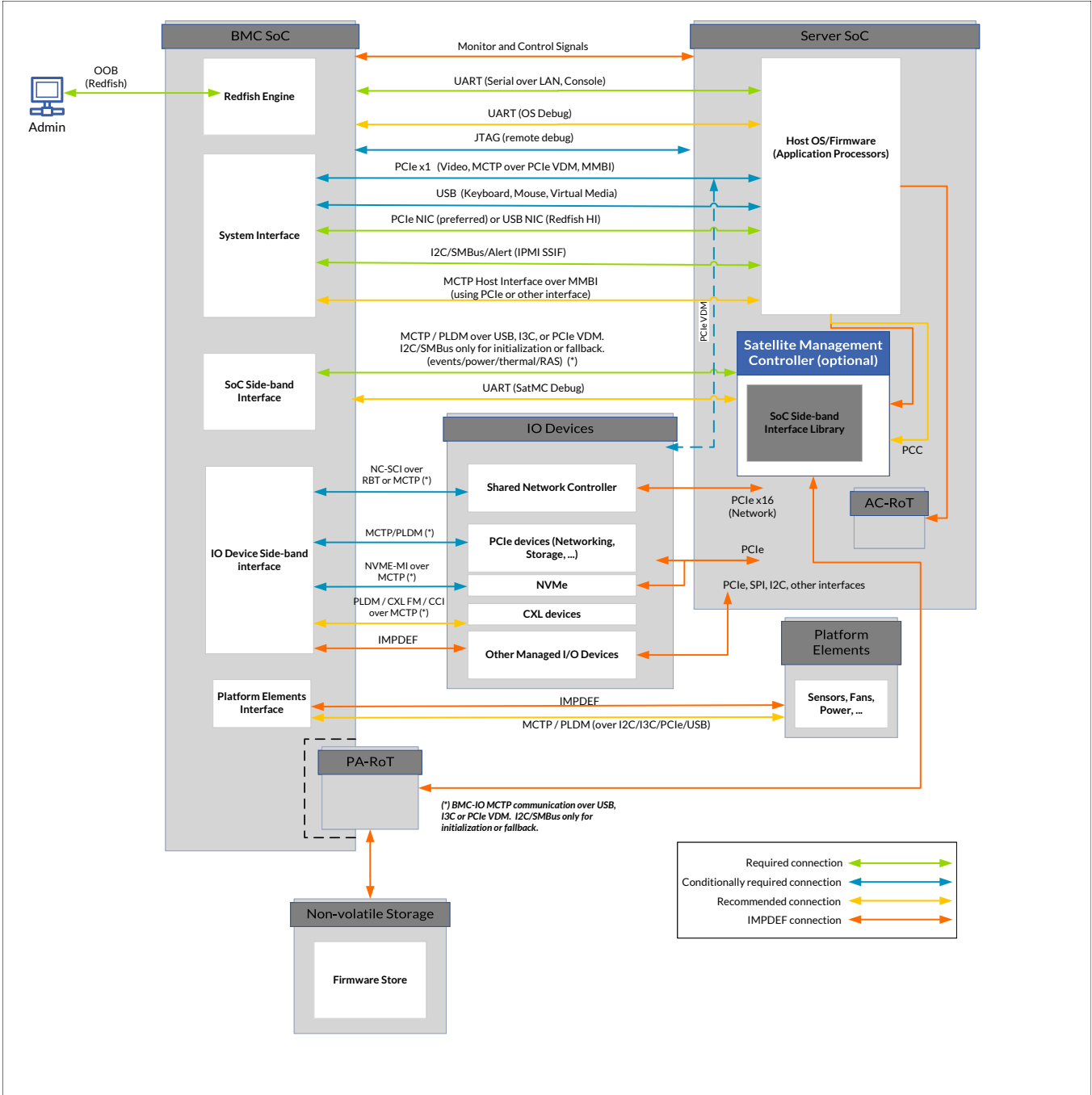


Figure 7: Server management interfaces for Level M5a

2.6.1 SoC-BMC interface

R For Level M5a-based server systems, the requirements and recommendations extend those of the level M4.

I It is recommended that the SatMC includes a BSA [20] compliant UART connection to the BMC for the purpose of remote SatMC debugging. Support for UART flow control is platform IMPLEMENTATION DEFINED. Using multiplexer of this UART with other UARTs connected to the BMC is platform IMPLEMENTATION DEFINED.

2.6.1.1 Host SoC in-band interface

I For Level M5a based server system, it is recommended that an additional higher bandwidth Host/SoC In-Band interface communication is implemented using the MCTP protocol over Memory-Mapped BMC Interface (MMBI) [[16]][17].

R_{RM5_IB_1} If the system implements MMBI, it must be compliant with the DMTF MMBI specification [16]. The MMBI physical interface may exist over PCIe MMIO or other interfaces, as supported by [16].

R_{RM5_IB_2} If the Host/SoC In-Band interface uses the MCTP protocol, then the interface must be discoverable through a mechanism defined in the MCTP Host Interface Specification [15].

2.6.1.2 Host SoC side-band interface

R_{M5_SB_1} For Level M5a based server systems, the physical and data-link layer methods for MCTP communication are defined by one of the following higher bandwidth transport options:

- MCTP over USB binding specification [51], or
- MCTP over I3C binding specification [40],
- MCTP over PCIe VDM binding specification [18]

I If MCTP over PCIe VDM is used, the PCIe bus can be either dedicated PCIe bus between the BMC and the SatMC, or shared with other PCIe links between the BMC and the SoC. It is recommended that a fallback side-band interface exists when the PCIe bus is not available, for example, if the PCIe bus is initialized late during boot, or if the link goes down.

I MCTP over SMBus/I2C [39] can be supported only for initial discovery, or as a fallback if the higher bandwidth interface is not available.

I If MCTP over USB is used, the USB Host Controller is in the BMC, and the USB Endpoint device in the SatMC. For bi-directional communication, the BMC continuously polls for content from the SatMC endpoint, as described in [51].

2.6.1.3 PA-RoT and AC-RoT

I The SoC can contain an AC-RoT responsible for actively monitoring the state of the firmware/software executing upon the SoC, and detecting any ongoing attack on the SoC.

I If a PA-RoT is present in the platform, the SoC may have an interface to it. The PA-RoT can be either a discrete system component or integrated in the BMC.

For guidance on the PA-RoT and AC-RoT responsibilities see Section G.5.

2.6.2 BMC-IO device interface

R For Level M5a-based server systems, the requirements and recommendations extend those of the level M4.

R_{M5_IO_1} For Level M5a based server systems, the physical and data-link layer methods for MCTP communication are defined by one of the following higher bandwidth transport options:

- MCTP over USB binding specification [51], or
- MCTP over I3C binding specification [40],
- MCTP over PCIe VDM binding specification [18]

I For details on how PCIe devices are managed out-of-band using MCTP and other protocols, see [52].

I If MCTP over PCIe VDM is used, and the PCIe bus is shared with other PCIe links between the BMC and the SoC, it is recommended that a fallback BMC-IO interface exists when the PCIe bus is not available, for example, if the PCIe bus is initialized late during boot, or if the link goes down.

- I MCTP over SMBus/I2C [39] can be supported for initial discovery, or as a fallback if the higher bandwidth interface is not available or not supported on older devices.
- I There are other use cases of SMBus/I2c communication between the BMC and IO devices. For example, OCP Secure Firmware Recovery [53] relies on SMBus/I2C as the firmware recovery protocol interface, and while that protocol does not require or depend on MCTP, it can co-exist with MCTP on the same physical bus.

2.6.3 BMC-platform elements interface

- R For Level M5a-based server systems, the requirements and recommendations extend those of the level M4.
- I When MCTP is used for BMC to platform elements communication, it is possible to use several physical and data-link layers, including I2C/SMBus [39], I3C [40], USB [51], and PCIe VDM [18]. Usage of slower busses such as I2C/SMBus is not recommended when high bandwidth communication is needed.
- I If a PA-RoT is present in the platform, the BMC may have an interface to it. The PA-RoT can be either a discrete system component or integrated in the BMC.

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

- R For Level M5a-based server systems, the requirements and recommendations extend those of the level M4.
- R_{M5_00B_1} If the server platform supports user-accessible BIOS settings configuration, the BMC must support the Redfish BIOS settings resource.
- I Platforms may support multiple interfaces to configure the BIOS settings, such as Redfish, local BIOS/UEFI setup, and other tooling. The priority and synchronization across these interfaces are IMPLEMENTATION DEFINED. For example, a platform may include an IMPLEMENTATION DEFINED method to enable/disable the ability to configure BIOS settings locally, and only allow Redfish configuration.
- I Redfish settings resource represents the future intended state of a resource. Some resources have properties that can be updated, and the updates take place immediately. However, some properties need to be updated at a future point in time, such as after a system reset. SBMR recommends using Redfish settings resource for BIOS configuration.
- I Redfish Resource and Schema Guide [30] provides information on how to use the Redfish interface, targeted at consumption of the interface.

2.6.5 SPDm over MCTP for BMC and side-band devices

- R For Level M5a-based server systems, the requirements and recommendations are the same as for M4-based server systems.

2.6.6 Host-to-SatMC interface

- R_{M5_HS_1} If the Host-to-SatMC interface uses a PCC mailbox, that mailbox must use PCC subspace structures of type 3 and type 4, and must carry MCTP traffic as defined by the MCTP over PCC binding specifications [54].
- R_{M5_HS_2} If the Host-to-SatMC interface uses the MCTP protocol, then the interface must be discoverable through a mechanism defined in the MCTP Host Interface Specification [15].
- I For Level M5a based server system, it is recommended that the Host-to-SatMC communication is performed using the MCTP protocol over a PCC mailbox composed of type 3 and type 4 PCC subspace structures [7].

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
IPMI	IPMI_8
RAS	M1_RAS_1
RAS	M1_RAS_2

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
RAS	M2_RAS_1
RAS	M2_RAS_2

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_3
PCIe	M21_PCI_1
USB	M21_USB_1
IPMI	M21_IPMI_1
IPMI	M21_IPMI_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
SPDM	M3_SPDM_2
RAS	M3_RAS_1

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
Side-Band	M4_SB_1
BMC-IO	M4_IO_1
BMC-IO	M4_IO_2
BMC-IO	M4_IO_3

2.7.6 SBMR Level M5a checklist

In addition to the SBMR Level M4 rules in Section [2.7.5](#), the following additional rules are required.

Category	Rule ID
In-Band	M5_IB_1
In-Band	M5_IB_2
Side-Band	M5_SB_1
BMC-IO	M5_IO_1
OOB	M5_OOB_1
Host-to-SatMC interface	M5_HS_1
Host-to-SatMC interface	M5_HS_2

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, as well as Arm.

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

B IPMI

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

B.1 Standard IPMI commands

The following are the IPMI commands defined by [10] which are required by this specification.

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 [10]

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.

U Implementations may limit the IPMI boot device selection command to reasonable boot targets that are supported by the system, such as PXE, HDD, CD/DVD, or USB. Support for special-purpose boot targets, such as boot from diagnostics partition, is not required.

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 [10]
- IPMI Get System Boot Options Command [10]

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

R_{IPMI_8}

The Redfish in-band Host Interface includes an optional feature to bootstrap temporary Redfish service host accounts using some IPMI commands. If present, these commands must follow the definitions in version 1.30 or newer of the Redfish Host Interface Specification [27].

B.1.5.1 IPMI commands

- IPMI Get Manager Certificate Fingerprint Command [27]
- IPMI Get Bootstrap Account Credentials [27]

B.1.5.2 Redfish properties

I

CredentialBootstrapping property defined in the HostInterface Redfish Schema [9] [55]. 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 [9] [55].

B.1.6 IPMI support verification

S

A script to verify the basic remote IPMI functionality is [available here](#). The Arm SBMR Architecture Compliance Test Suite (SBMR-ACS) also provides similar tests for remote and local IPMI functionality, and is [available here](#).

B.2 Arm standard IPMI commands

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

B.2.1 General IPMI commands format

The common components of IPMI message as defined by [10] 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 [10]. 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 11 lists Arm standard IPMI commands that are defined in SBMR.

Table 11: List of Arm standard IPMI commands

Command	NetFn	Command Code	Definition
Send Platform Error Record	2Ch	01h	Section C.1.2
Send Boot Progress Code	2Ch	02h	Section F.1

B.3 IPMI specification clarifications and corrections

The following section lists corrections and clarifications to the IPMI specification [10] that directly impact IPMI implementation on Arm-based SBMR systems, including complete support for IPMI SSIF interface. These corrections are listed here in Table 12 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 12: SBMR deviations from the IPMI specification

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						
Complete address (7)	R/W=0 (1)	SMBus CMD =07h	Length	IPMI Data	[PEC]	Complete 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 14:

Table 14: IPMI SSIF multi-part Start transactions

Byte 1		Byte 2	Byte 3	Byte 4		Byte 5	Byte 6+	
Completer 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

The NetFun code is "0x2C" and the first byte of IPMI request data is 0xAE to indicate that the IPMI commands are defined by this specification.

Multi-part Middle transactions look like this Table 15:

Table 15: IPMI SSIF multi-part middle transactions

Byte 1		Byte 2	Byte 3	Byte 4+	
Completer 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 16:

Table 16: IPMI SSIF multi-part end transactions

Byte 1		Byte 2	Byte 3	Byte 4+	
Completer 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 M1

I Figure 8 shows a conceptual illustration of IPMI based in-band, SoC side-band, and out-of-band RAS interfaces for Level M1-based server systems.

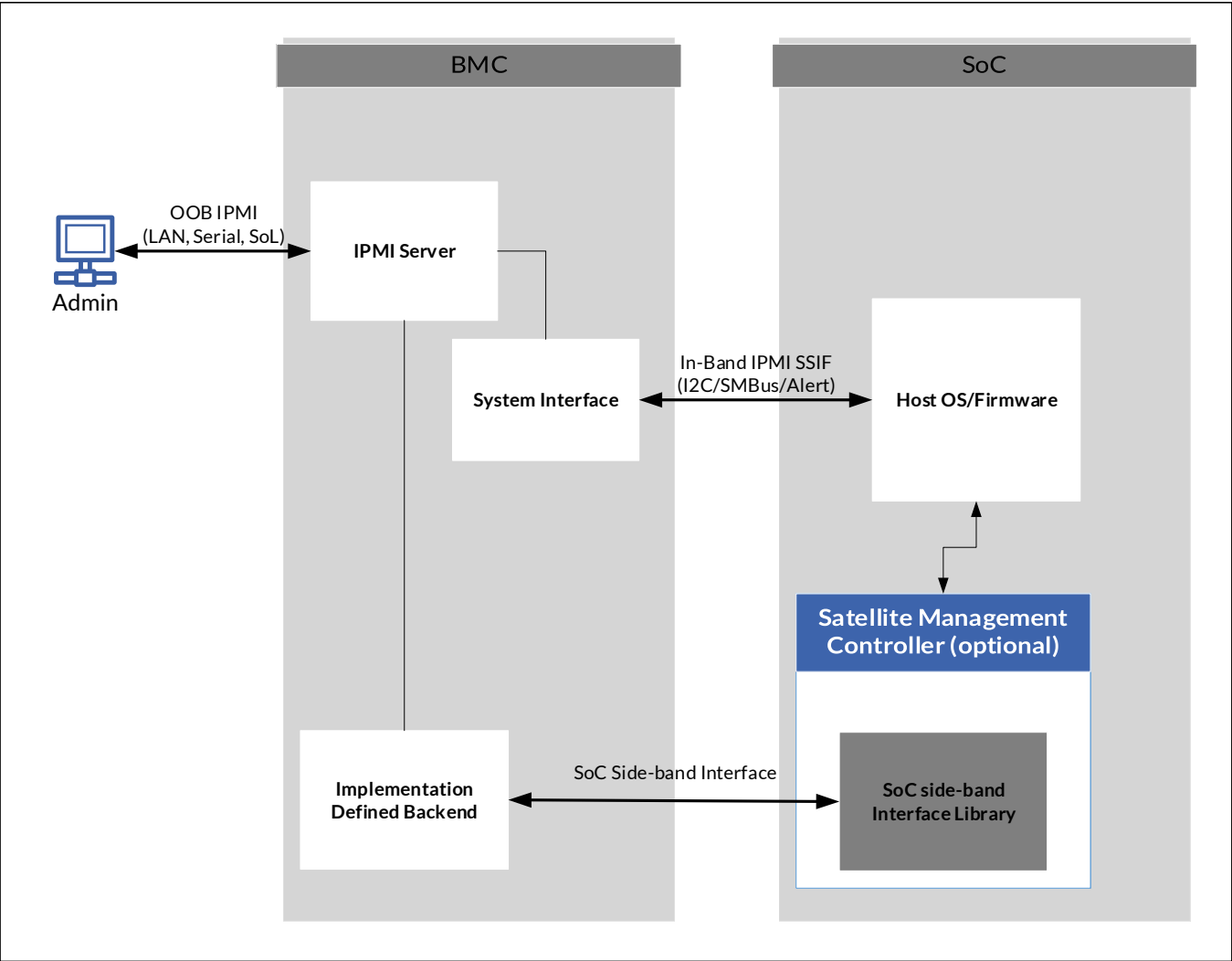


Figure 8: IPMI based RAS Interfaces

C.1.1 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.1.2.
- 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 9 illustrates the overview of RAS Events interaction with the event receiver and RAS Manager through SMBus System Interface (SSIF).

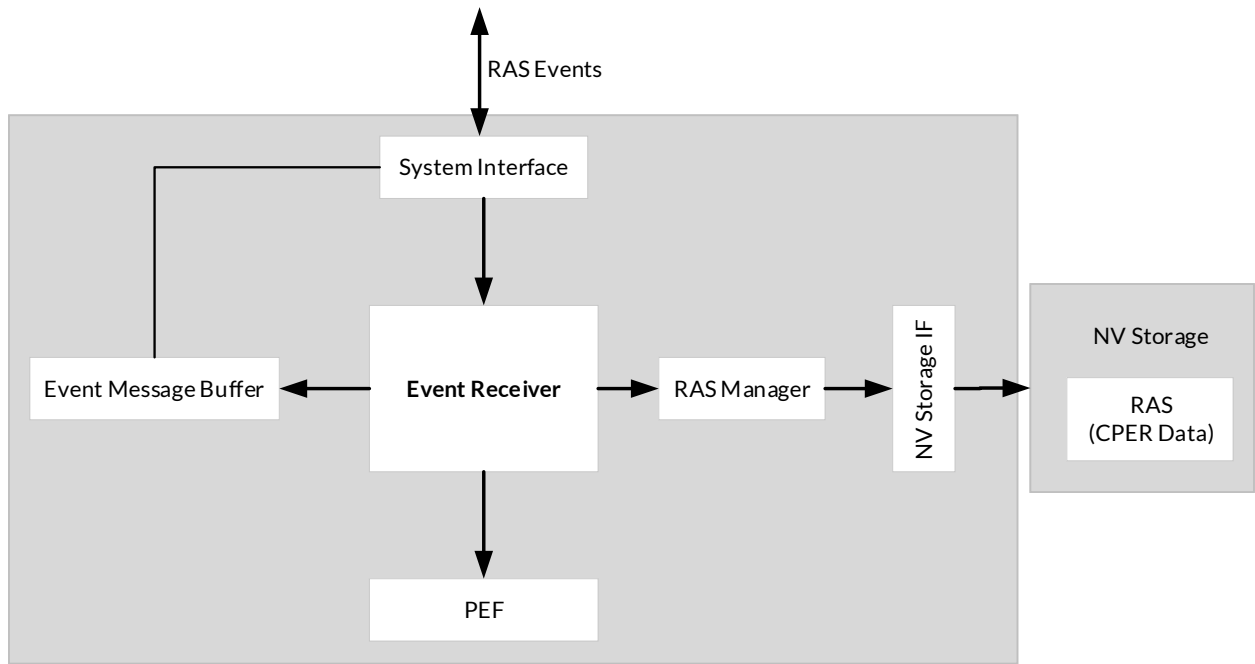


Figure 9: IPMI based RAS Event Receiver

U Figure 9 represents a conceptual illustration of the way that RAS event messages can be handled by a BMC 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 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 [8] (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 [8] (UEFI § N).

C.1.2 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.1.2.1 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, see Section B.3.

C.1.3 SoC side-band interface

I RAS error records can be generated in the host OS or the firmware, then transferred over to the 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.1.4 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.2 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 10 shows a conceptual illustration of these interfaces for RAS.

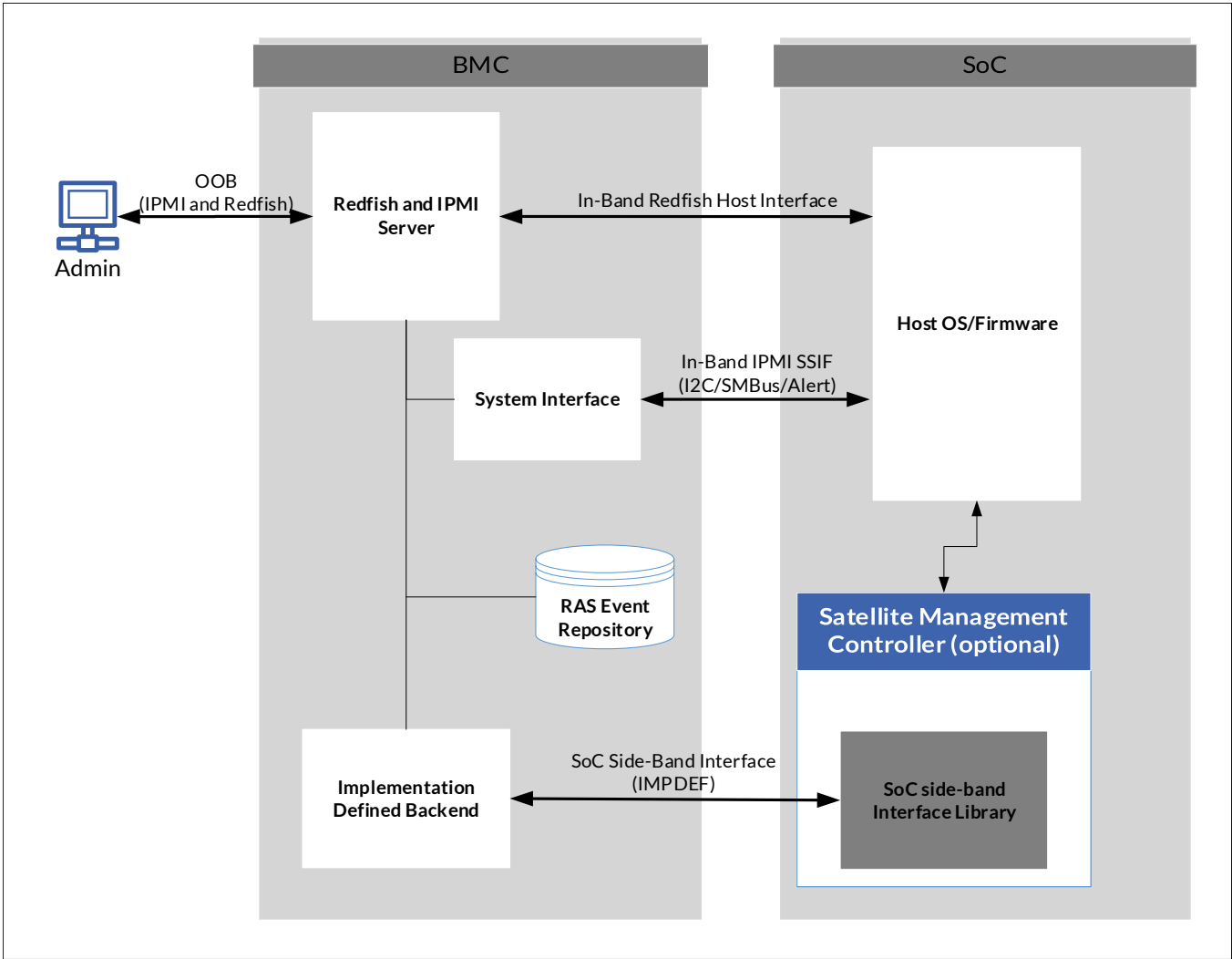


Figure 10: Redfish and IPMI based RAS Interfaces

C.2.1 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.1 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.2.2 RAS Redfish message format

R_{M2_RAS_2} The Redfish model for extracting Platform Error Records is defined in DMTF Redfish Schema [9], 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 schema.
- **AdditionalDataSizeBytes**: Size of the diagnostics binary file in bytes
- **DiagnosticData**: Optional Base64 representation of the platform error record binary. Subject to the constraints defined in the LogEntry schema.
- **DiagnosticDataType**: The type of the diagnostics binary file retrieved from **AdditionalDataURI** or available in **DiagnosticData**. For platform error records, this can be **OS**, **PreOS**, **CPER**, **CPERSection**, or **DEM**. 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.
- **CPER.SectionType**: The type of the CPER section.
- **CPER.NotificationType**: The notification used for the CPER record.

C.2.3 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.2.4 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.3 Level M3, M4, and M5a

- 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 11 shows a conceptual illustration of these interfaces for RAS.

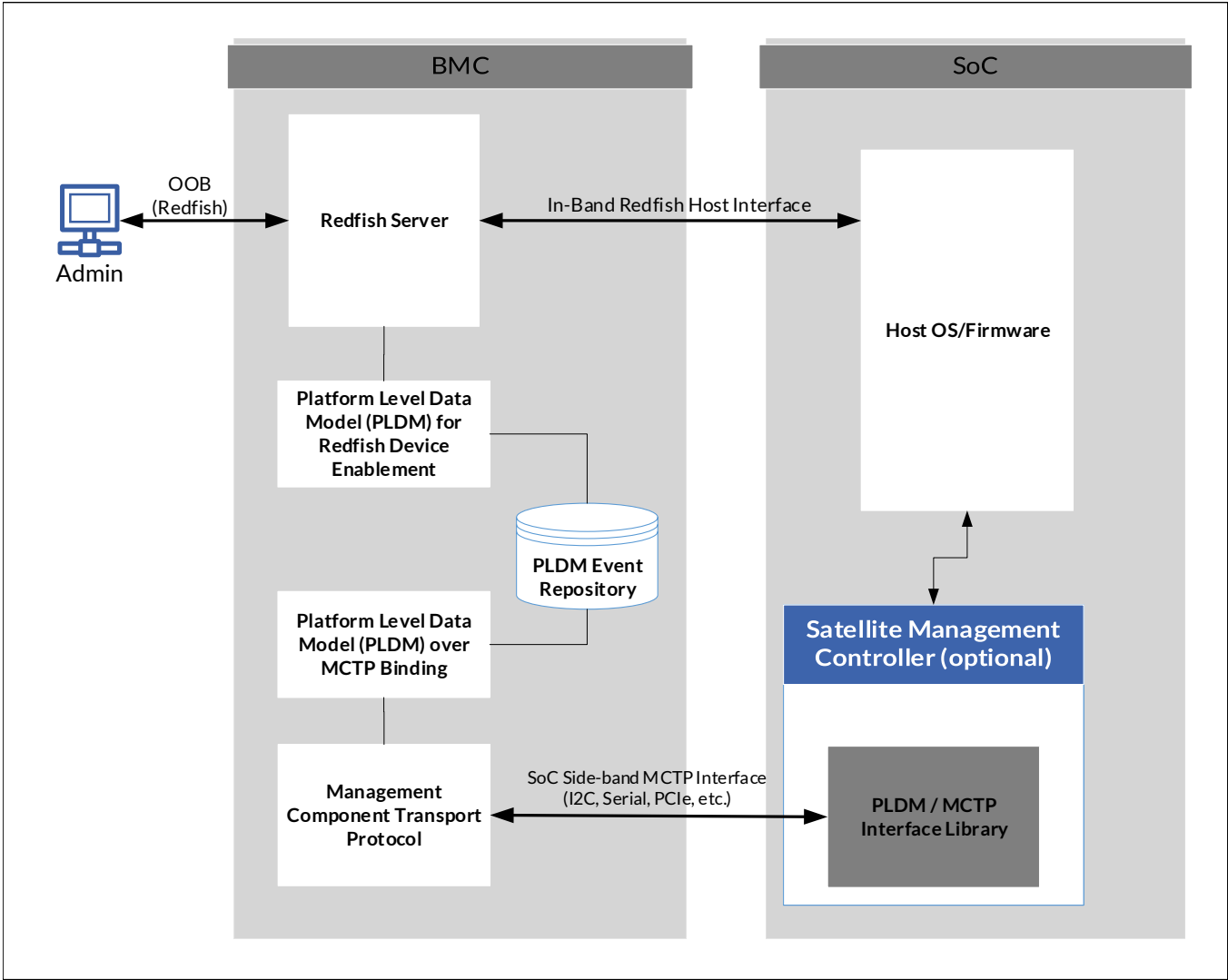


Figure 11: Redfish/PLDM/MCTP based RAS Interfaces

C.3.1 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.3.2 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 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 Levels M3, M4, and M5a based server systems.
- I SBMR recommends Platform Level Data Model (PLDM) as the SoC side-band message definition and data layer interface for the Levels M3, M4, and M5a 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 SatMC to the

BMC.

- I A new event class `CPEREvent` is defined to enable this feature. SBMR recommends using the `CPEREvent` value **07h**, which is defined in Table 11 of DSP0248 v 1.3.0 or newer [32]. The format of `CPEREvent` is defined in Table 27 of [32].
- I SBMR previously defined `oemEvent` value of **FA** for use as a temporary `CPEREvent` event class until an industry standard value is defined. Some existing implementations are already using the `oemEvent` value. SBMR recommends using the standard value defined in [32] v 1.3.0 where possible.
- I 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 `GetNextPart` to initiate the transfer. In response, the SatMC 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, see [32].

C.3.2.1 RAS PLDM message flows examples

- U Figure 12 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 13 shows an example flow that SatMC use asynchronous method to send small event to BMC.
- U Figure 14 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 15 shows an example flow that BMC use polling method to receive a large multi-part event.
- U Figure 16 shows an example flow that BMC use polling method to receive a small event.
- U Figure 17 shows an example flow that SatMC reports an empty event queue when BMC try to poll an event back.

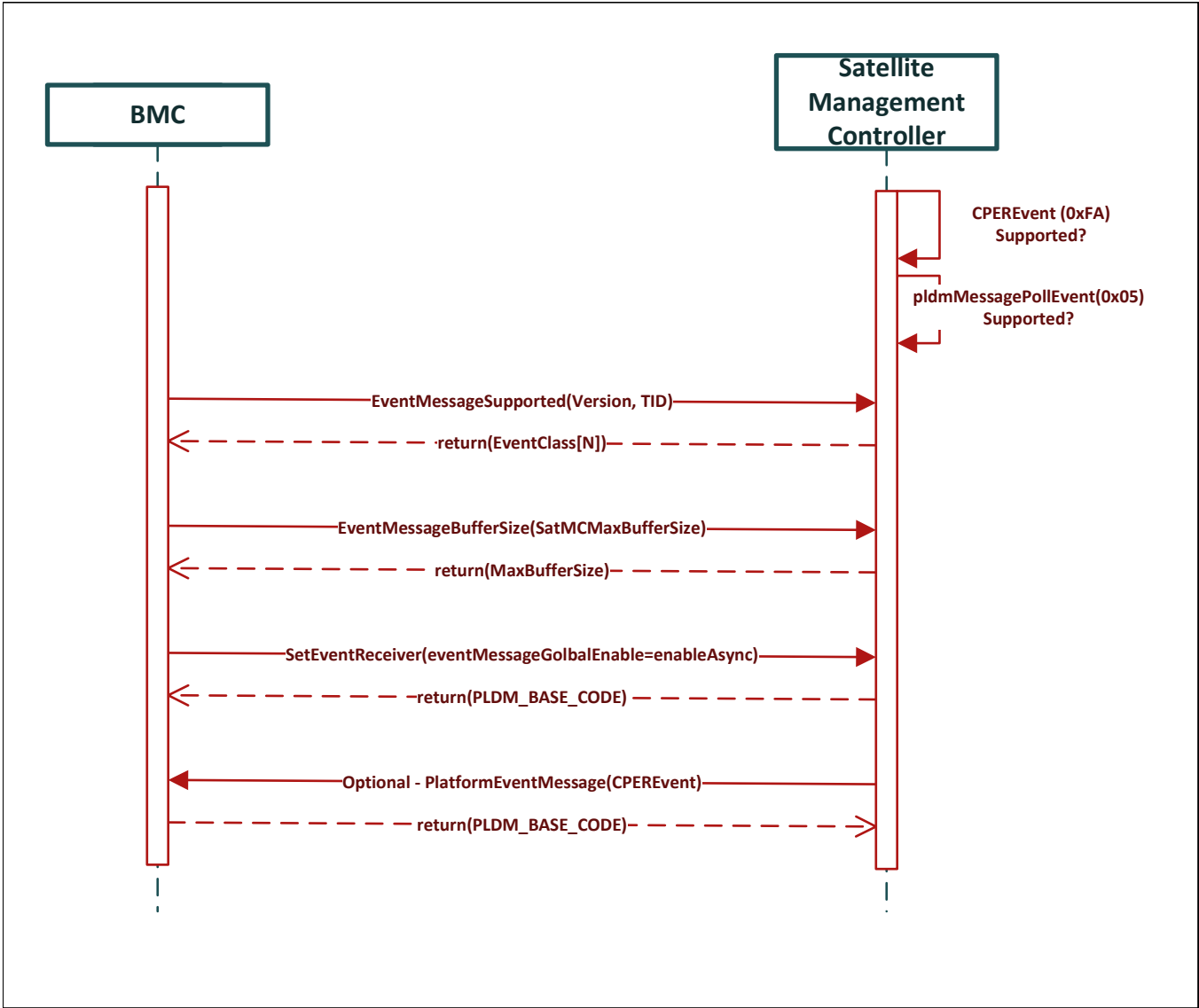


Figure 12: RAS side-band flow – boot initialization

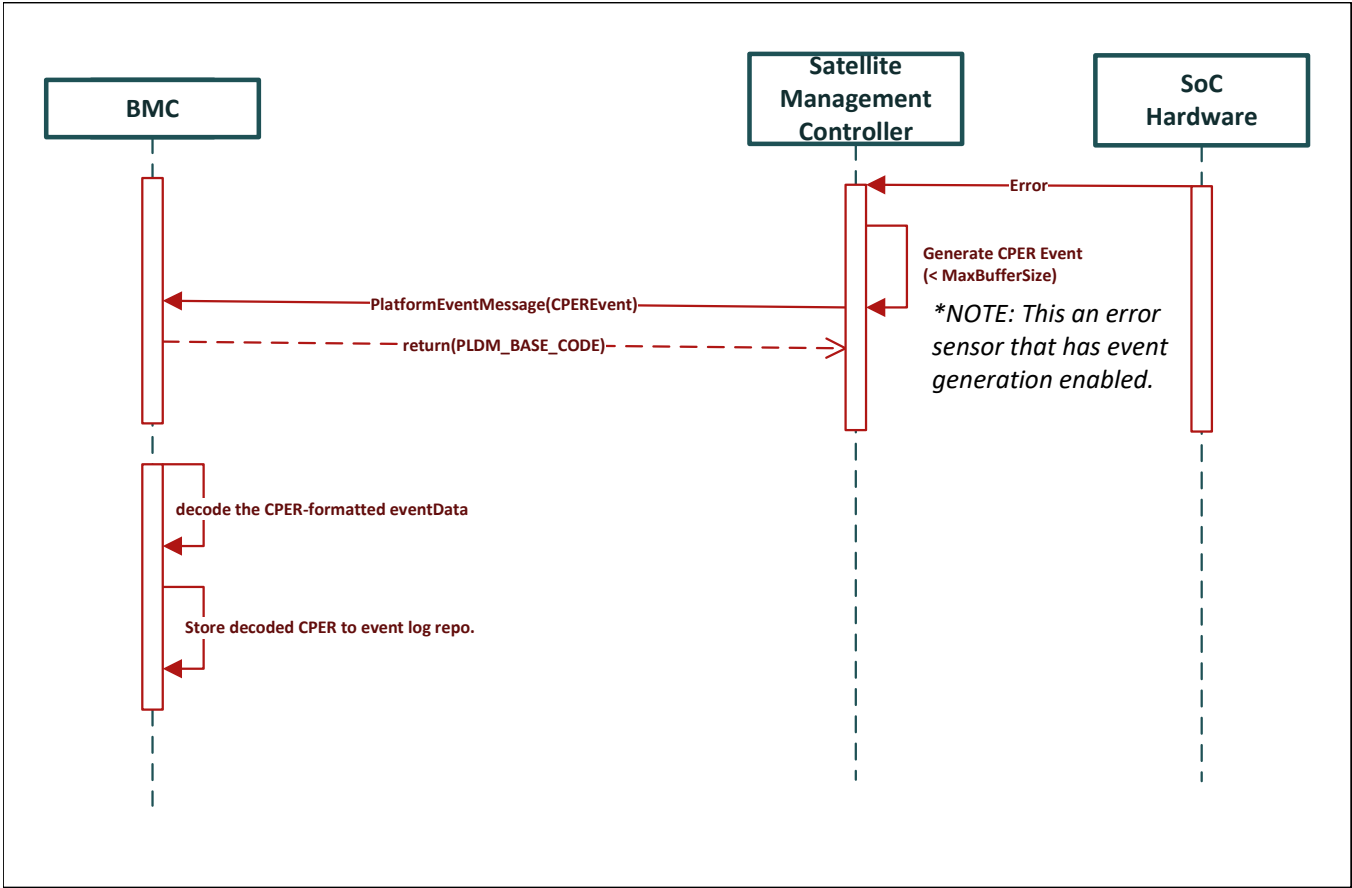


Figure 13: RAS side-band flow – Asynchronous small event

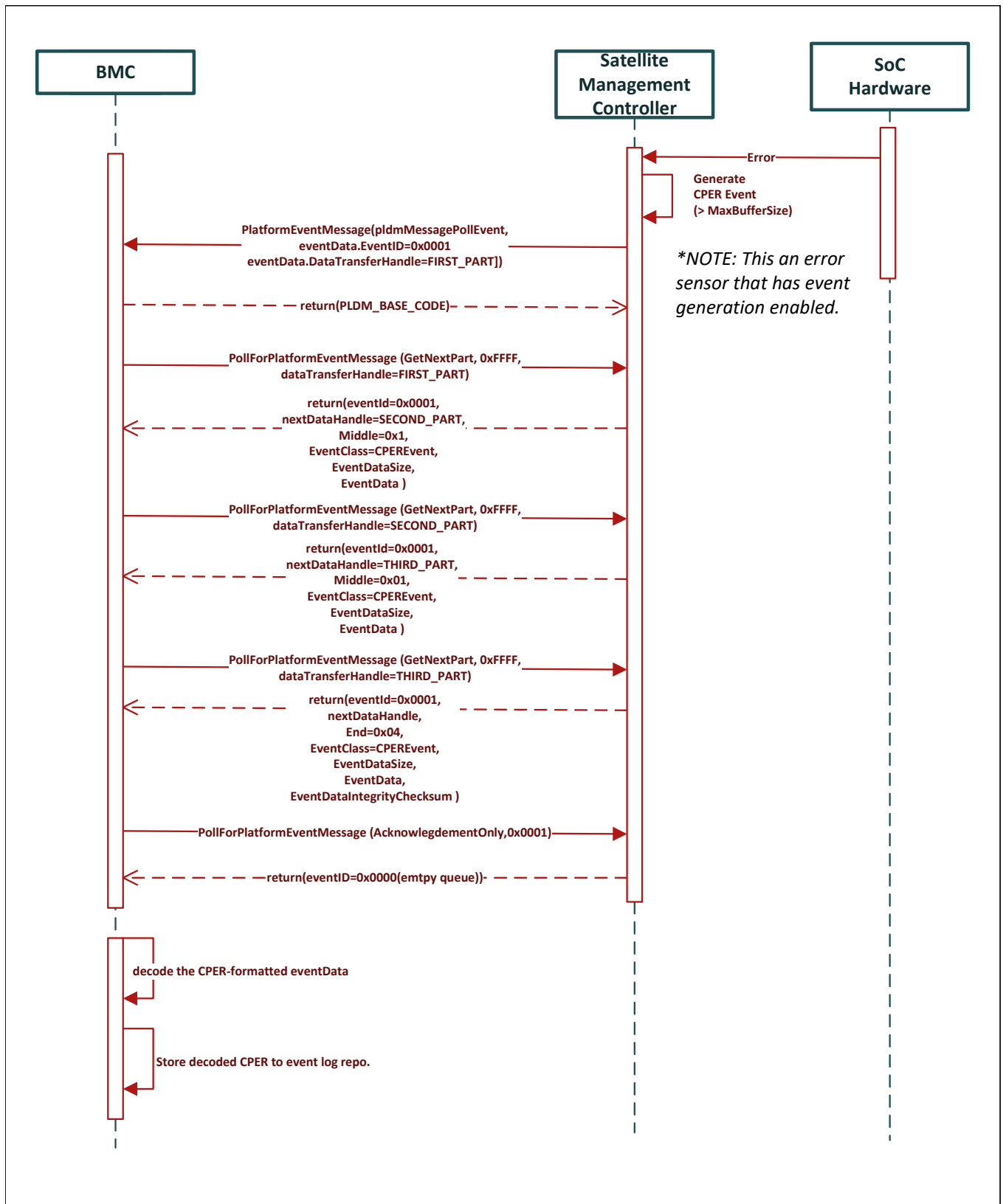


Figure 14: RAS side-band flow – SatMC async notification for BMC to switch to poll for large event

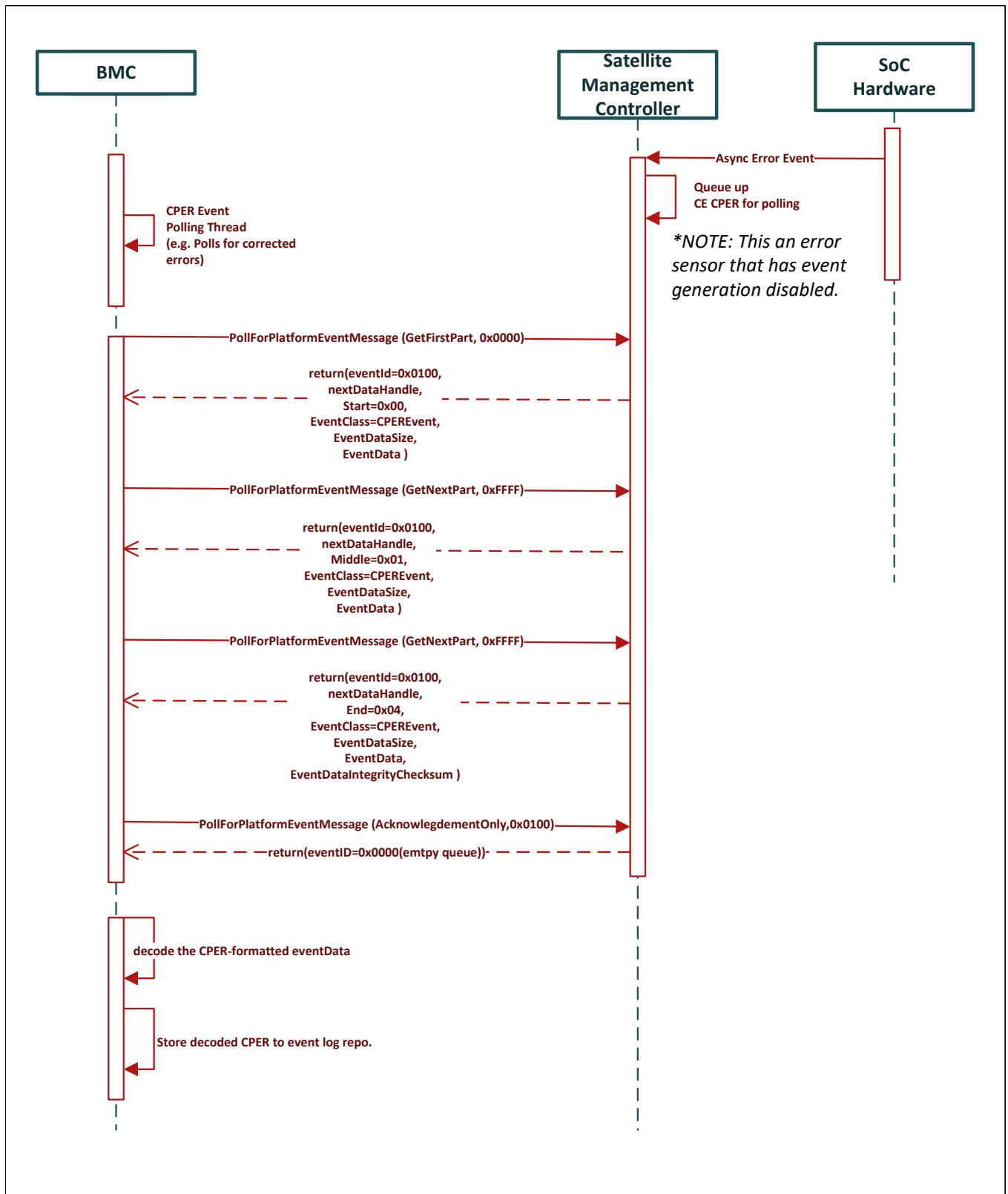


Figure 15: RAS side-band flow – BMC polling to receive large event

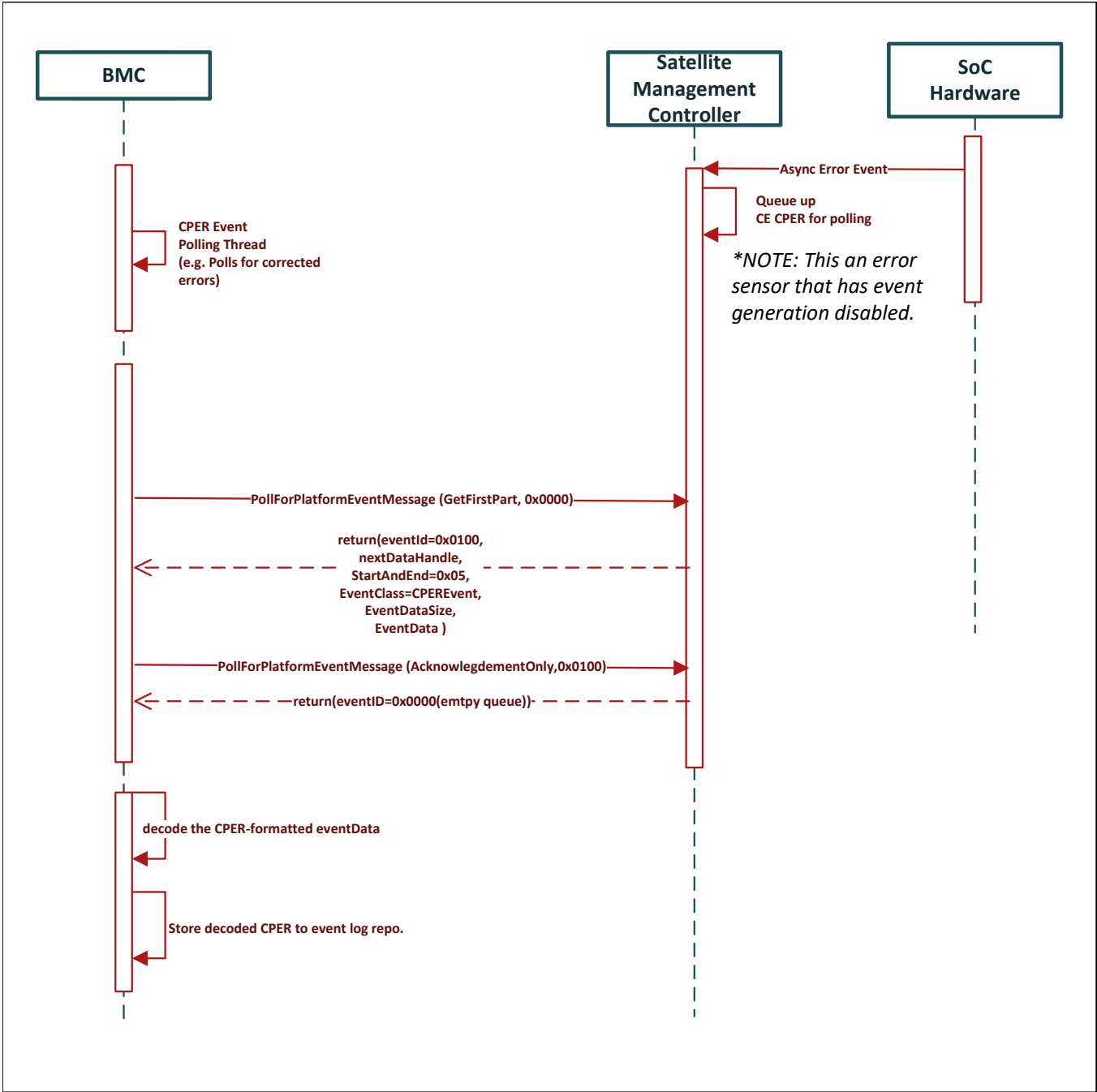


Figure 16: RAS side-band flow – BMC polling to receive small event

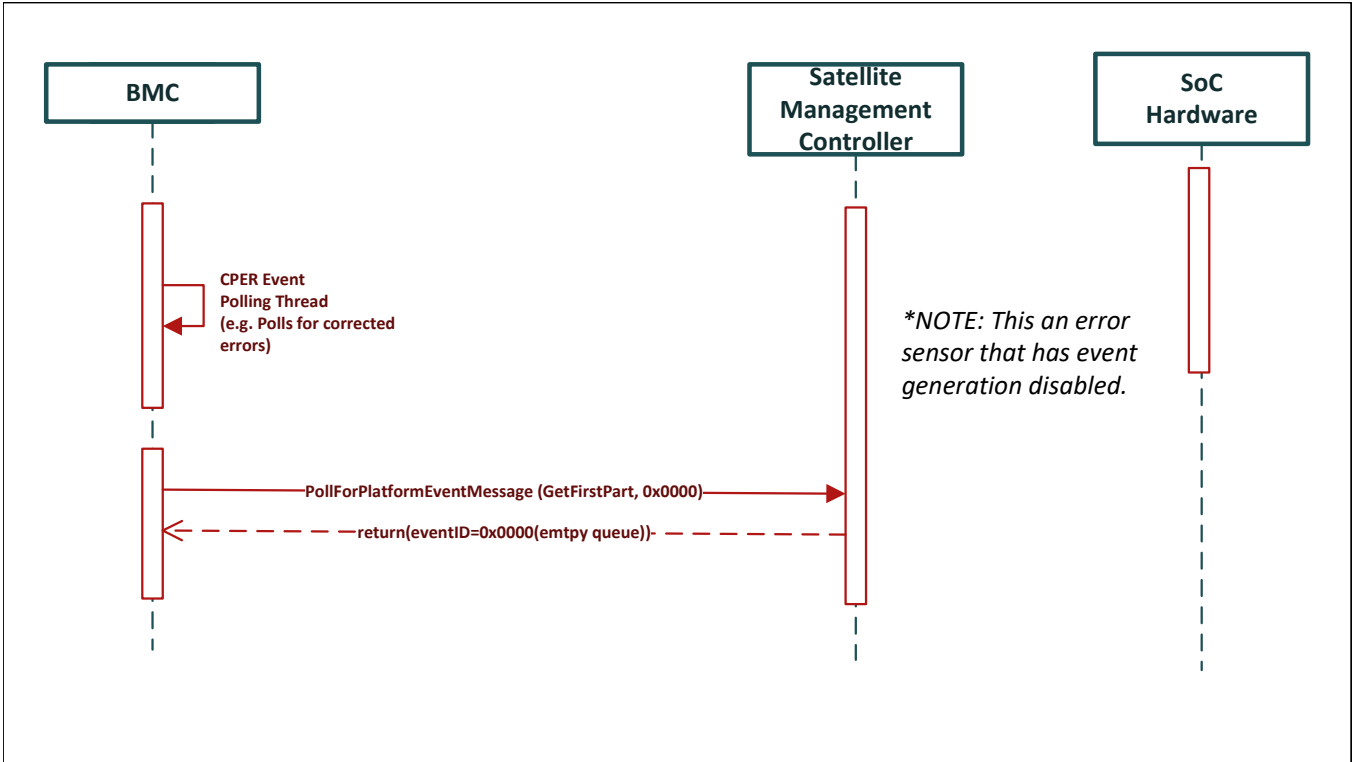


Figure 17: RAS side-band flow – SatMC reporting of empty event queue

C.3.3 Out-of-band interface

- I SBMR recommends a Redfish based tool to extract the stored RAS error records in a CPER format from the PLDM event repository.
- U When the BMC receives the CPER binary from the SatMC and stores it in its local repository, the BMC generates corresponding event log entry to Redfish event log repository as explained in Section C.2.2. 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. When found, the tool can download the CPER file pointed to by the AdditionalDataURI property of the Redfish log entry for that error event. Alternatively, the tool can retrieve the DiagnosticData Base64 representation of the CPER binary. Figure 18 shows the CPER dataflow from SatMC to Remote Redfish base tool.

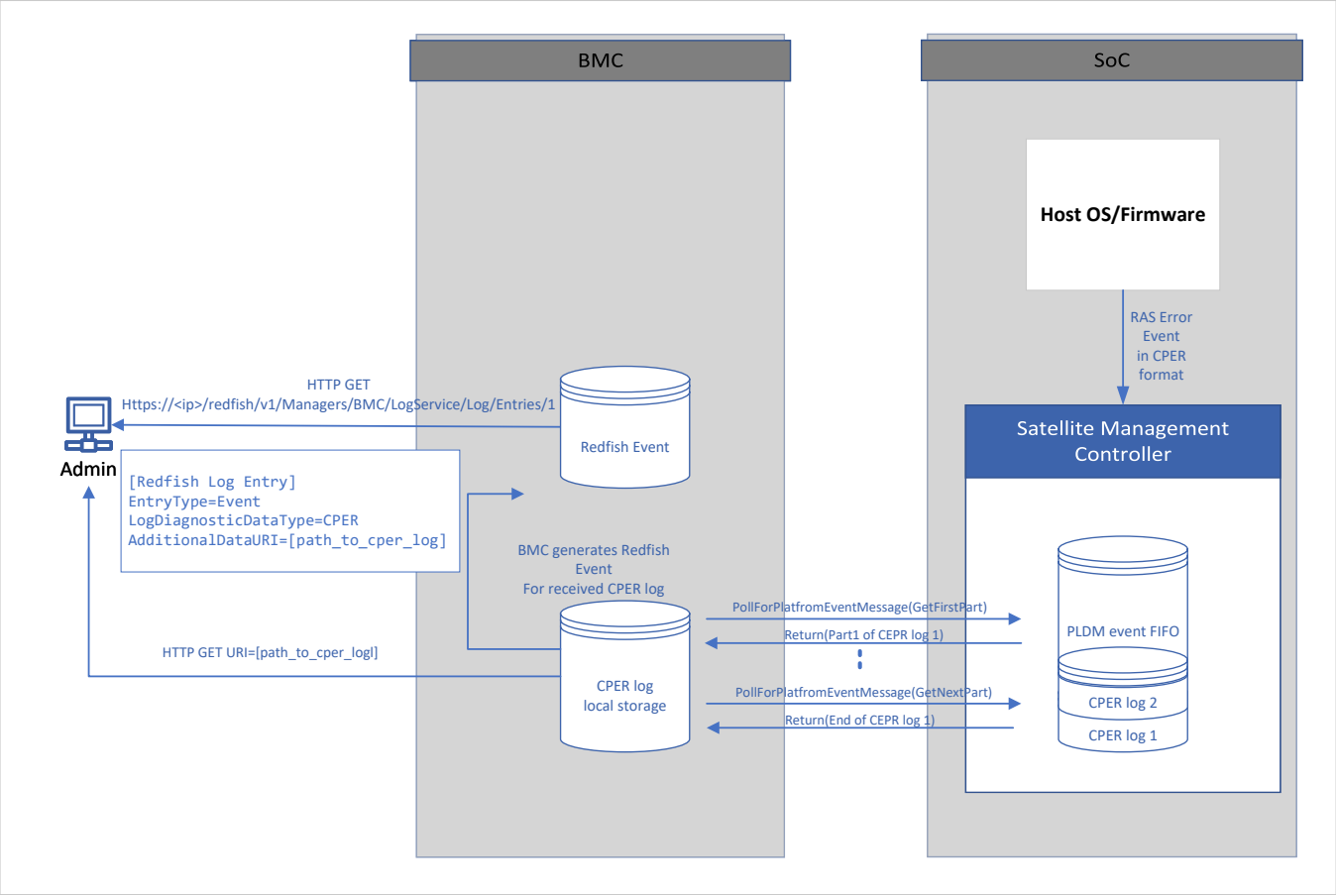


Figure 18: Out-of-band interface for RAS/CPER event log

D Platform monitoring and control

D.1 Background

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, see the IPMI Specification [10].

Sensor data records (SDRs)

I SBMR recommends SDR Type 01h, Full Sensor Record, to describe the managed entities. For more details, see the IPMI Specification [10].

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 [10] for managed entities. For more details on the Sensor Type values, see [10] (IPMI § Table 42-3).

For a list of sensor unit codes, see [10] (IPMI § Table 43-15).

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 (for example, pre-EFI firmware), and SoC Management Software (for example, 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, see [10] (IPMI § Table 43-13).

D.3 Redfish schema to monitor and control managed entities

I SBMR recommends the use of the Redfish schema for sensor as defined by [9] [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)
SetNumericEffectorEnable	C	SatMC	To be implemented when SatMC
SetNumericEffectorValue	C	SatMC	has numeric effector(s)
GetNumericEffectorValue	O	SatMC	
SetStateEffectorEnables	C	SatMC	To be implemented when SatMC
SetStateEffectorStates	C	SatMC	has state effector(s)

PLDM Platform command	M/C/O	Responder	Description
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 [32]

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
GetPDRRepositoryInfo	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.
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, see [32].

I

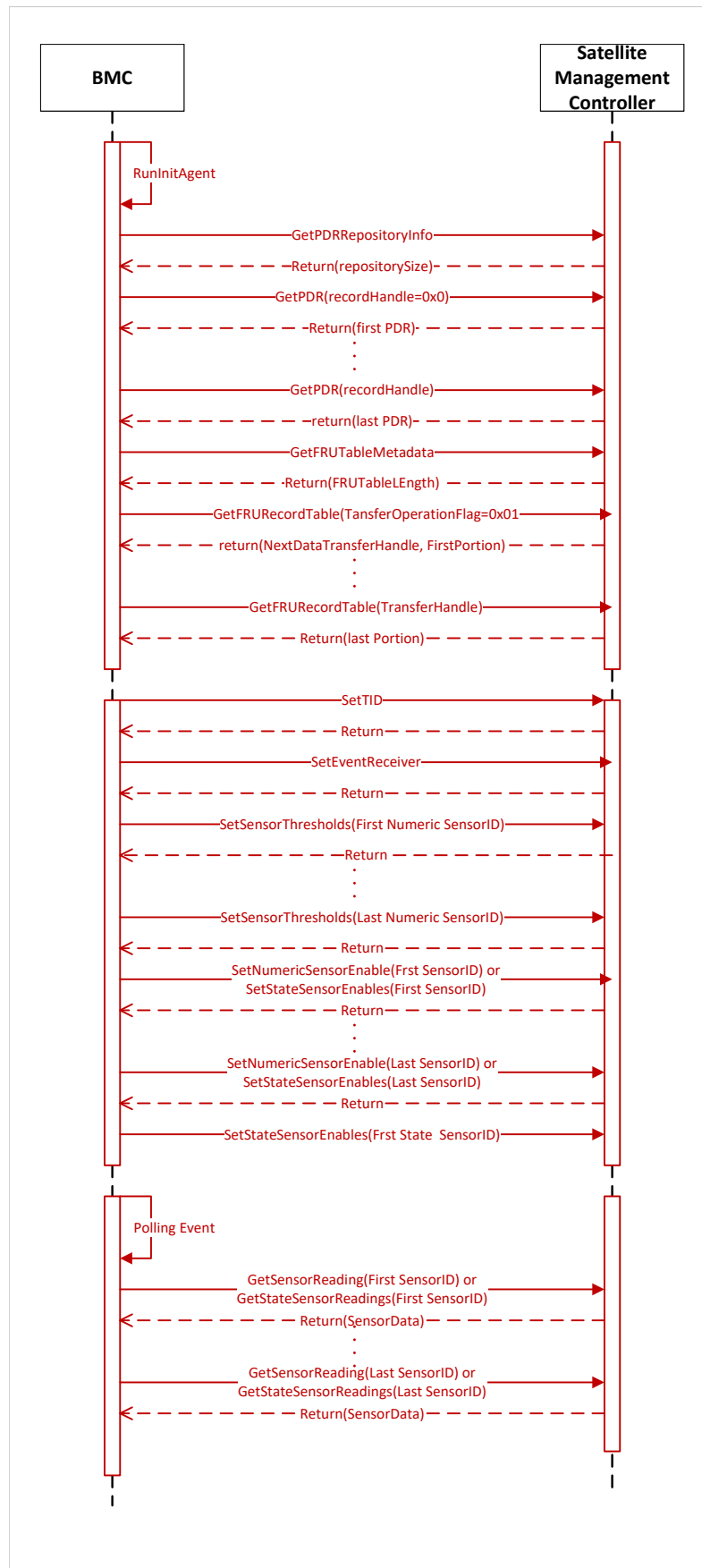
If PDRs are used, SBMR recommends the following types of Platform Descriptor Records (PDRs) in Table 23 to be implemented.

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 Effector PDR	O	SatMC	If PDRs are used, SatMC should implement this
Numeric Effector Initialization PDR	O	SatMC	PDRs if SatMC supports Numeric Effector(s) and BMC has no knowledge of accessing these sensors.
State Effector PDR	O	SatMC	If PDRs are used, SatMC should implement this
State Effector Initialization PDR	O	SatMC	PDRs if SatMC supports State Effector(s) and BMC has no knowledge of accessing these sensors.

U

The flowchart in Figure 19 shows how the BMC uses PLDM commands and PDRs to retrieve sensor / platform monitoring data from a SatMC.

**Figure 19: BMC PLDM/PDR monitoring 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 [33] 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, Processor	288, Processor Power State	The Set ID value of the state sensor PDR should be 288 if the ACPI power state in [33] table 10 can be applied to the CPU/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 [33] table 10 can be applied to the CPU/SoC in system.
DIMM Group N max. temp	Numeric	66, Memory module	N/A	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 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 Background

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, M5a

- U This section describes a reference solution for implementing BMC remote debug using [OpenOCD](#) for SBMR Levels M1, M2, M2.1, M4, M4, and M5a 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 Figure 20 shows a reference implementation of remote debug feature using GNU MCU Eclipse plugin and OpenOCD using JTAG interface.

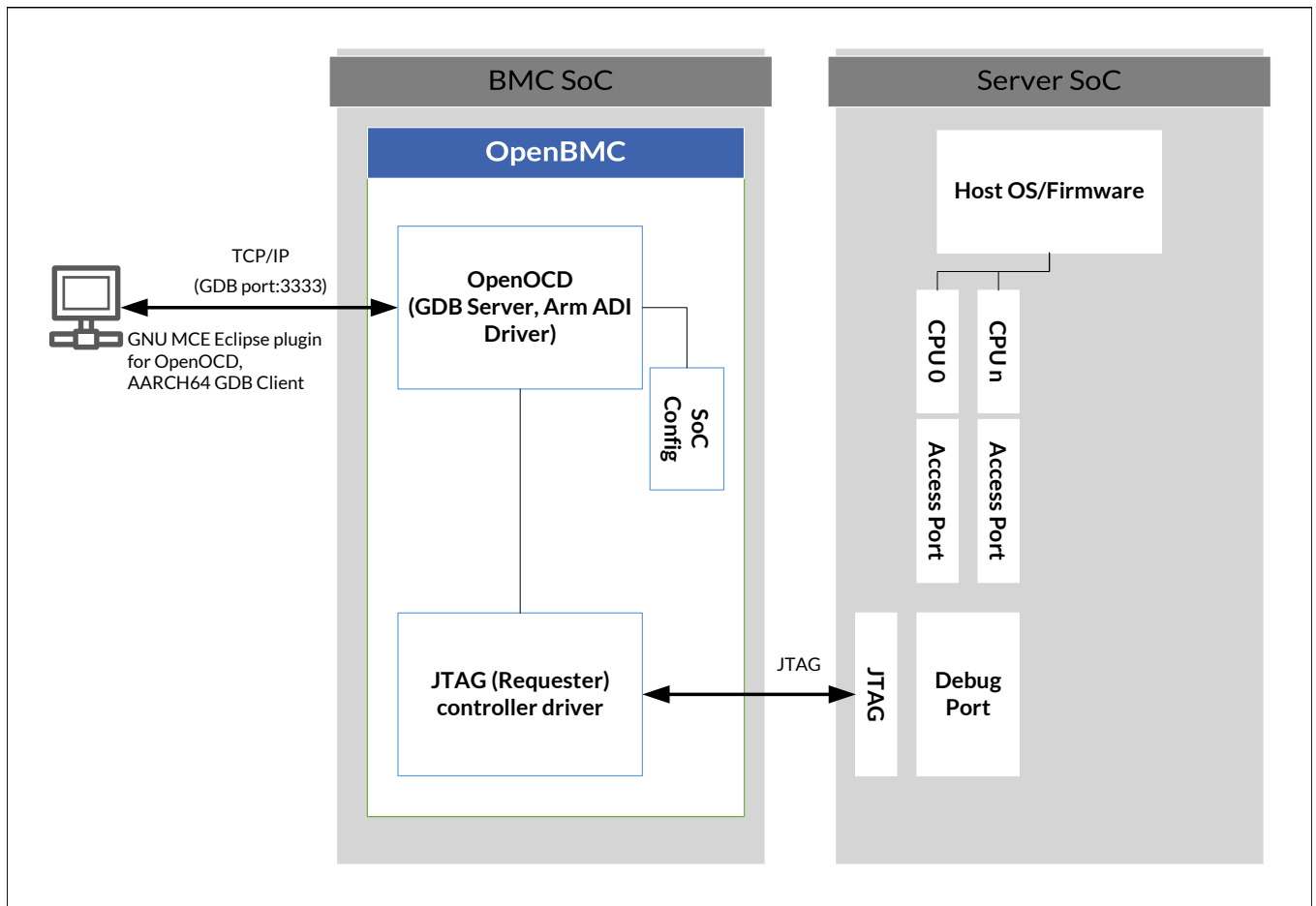


Figure 20: Reference implementation of remote debug.

- U Client running on the remote machine connected to OpenOCD GDB Server running on the BMC. OpenOCD includes a JTAG controller (requester) 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, see OpenOCD user guide [56].
- 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

R 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)

R 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)

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

Request Data

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

R The format of the Boot Progress Code data follows the definitions of `EFI_STATUS_CODE_TYPE` and `EFI_STATUS_CODE_VALUE`, as defined in the PI Specification [57] (PI § Vol 1-4.7, PI § Vol 2-14.2, PI § Vol 3- 6).

R If the PI Specification [57] adds new definitions, such as new classes, sub-classes, or operations, it is assumed that the values are valid for usage in this specification.

R The boot progress code is composed of `EFI_STATUS_CODE_TYPE`, `EFI_STATUS_CODE_VALUE` and Instance:

- `EFI_STATUS_CODE_TYPE` (4 Bytes)
 - 32-bit field matching `EFI_STATUS_CODE_TYPE` as defined by the PI Specification [57] (PI § Vol 1-4.7, PI § Vol 2-14.2, PI § Vol 3- 6). The field is composed of the following sub-fields:
 - `STATUS_CODE_TYPE` (1 Byte)
 - * 0x01 = `PROGRESS_CODE`
 - * 0x02 = `ERROR_CODE`
 - * 0x03 = `DEBUG_CODE`
 - `STATUS_CODE_RESERVED` (2 Bytes)
 - * Reserved by PI Specification. set to 0x0000
 - `STATUS_CODE_SEVERITY` (2 Bytes)
 - * 0x40 = `ERROR_MINOR`
 - * 0x80 = `ERROR_MAJOR`
 - * 0x90 = `ERROR_UNRECOVERED`
 - * 0xa0 = `ERROR_UNCONTAINED`
- `EFI_STATUS_CODE_VALUE` (4 Bytes)
 - 32-bit field that follows the format of `EFI_STATUS_CODE_VALUE` as defined by the PI Specification [57] (PI § Vol 1-4.7, PI § Vol 2-14.2, PI § Vol 3- 6).
 - `EFI_STATUS_CODE_OPERATION` (2 Bytes)
 - * 0x0000-0x0FFF : Shared by all sub-classes in a class
 - * 0x1000-0x7FFF : Subclass Specific
 - * 0x8000-0xFFFF : OEM specific. 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)
 - `EFI_STATUS_CODE_SUBCLASS` (1 Byte)
 - * 0x00-0x7F : Defined or Reserved by PI specification
 - * 0x80-0xFF : Reserved for OEM use. 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)
 - `EFI_STATUS_CODE_CLASS` (1 Byte)
 - * 0x00 = `COMPUTING_UNIT`
 - * 0x01 = `PERIPHERAL`

- * 0x02 = IO_BUS
- * 0x03 = SOFTWARE
- * 0x04-0x7F : Reserved by the PI Specification
- * 0x80-0xFF : Reserved for OEM use. 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)
- Instance
 - Matches the Instance parameter of ReportStatusCode() PEI service and DXE Protocol interface, as defined by the PI Specification [57] (PI § Vol 1-4.7, PI § Vol 2-14.2, PI § Vol 3- 6).
 - This is an optional field. When unknown, the Instance value is 0.

F.2.1 IPMI progress code definition

I The IPMI boot progress code definition follows Section F.2, as defined in the PI specification [57].
 R Table 29 defines the layout of the IPMI boot progress code.

Table 29: IPMI progress code definition

Byte offset	Size (Bytes)	Description
0	1	STATUS_CODE_TYPE
1	2	STATUS_CODE_RESERVED
3	1	STATUS_CODE_SEVERITY
4	2	EFI_STATUS_CODE_OPERATION
6	1	EFI_STATUS_CODE_SUBCLASS
7	1	EFI_STATUS_CODE_CLASS
8	1	Instance

F.2.2 PLDM progress code definition

I The PLDM boot progress code definition follows Section F.2, as defined in the PI specification [57].
 R The boot progress code is encoded as a PLDM numeric sensor of uint64 sensorDataSize [32].
 R Table 30 defines the layout of the PLDM boot progress code.

Table 30: PLDM progress code definition

Byte offset	Size (Bytes)	Description
0	1	STATUS_CODE_TYPE
1	2	STATUS_CODE_RESERVED
3	1	STATUS_CODE_SEVERITY
4	2	EFI_STATUS_CODE_OPERATION
6	1	EFI_STATUS_CODE_SUBCLASS

Byte offset	Size (Bytes)	Description
7	1	EFI_STATUS_CODE_CLASS

I The progress code representation as a PLDM numeric sensor does not capture the Instance number. It is assumed that a single instance exists.

F.2.3 Example progress codes (IPMI, PLDM)

U The following are some examples of Boot Progress Codes that are based on standard Status Code values that are defined by [57].

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 send boot progress command:
 - 0x2C 0x02 0xAE 0x01 0x00 0x00 0x00 0x00 0x10 0x01 0x00 0x00
- PLDM raw numeric sensor reading:
 - 0x01 0x00 0x00 0x00 0x00 0x10 0x01 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 send boot progress command:
 - 0x2C 0x02 0xAE 0x01 0x00 0x00 0x00 0x00 0x10 0x10 0x0F 0x03 0x00
- PLDM raw numeric sensor reading:
 - 0x01 0x00 0x00 0x00 0x10 0x10 0x0F 0x03

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 send boot progress command:
 - 0x2C 0x02 0xAE 0x01 0x00 0x00 0x00 0x01 0x10 0x01 0x02 0x00
- PLDM raw numeric sensor reading:
 - 0x01 0x00 0x00 0x00 0x01 0x10 0x01 0x02

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 send boot progress command:
 - 0x2C 0x02 0xAE 0x02 0x00 0x00 0x00 0x03 0x10 0x05 0x00 0x02
- PLDM raw numeric sensor reading:
 - 0x02 0x00 0x00 0x90 0x03 0x10 0x05 0x00

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 send boot progress command:
 - 0x2C 0x02 0xAE 0x02 0x00 0x00 0x00 0x12 0x80 0x0C 0x02 0x04
- PLDM raw numeric sensor reading:
 - 0x02 0x00 0x00 0x90 0x03 0x10 0x05 0x00

F.2.4 Example boot progress codes (Redfish)

DMTF Redfish Schema [9] and [55] 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 8-byte hex values defined in Section F.2, not including the Instance number.

U

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

Example 1 - Host processor power-on initialization

(See Example 1 in Section F.2.3)

- 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`
- PLDM raw numeric sensor reading:
 - `0x01 0x00 0x00 0x00 0x00 0x10 0x01 0x00`
- Redfish JSON mockup when reading the Progress Code from the Redfish interface:

```
{
  "BootProgress": {
    "LastState": "OEM",
    "OemLastState" : "0x0100000000100100",
    "LastStateTime": "2020-03-13T04:14:13+06:00",
  },
}
```

F.3 Common boot progress codes

Table 36 and Table 37 describe some common combinations of Boot Progress Codes and Boot Error Codes that can be used. For the raw values of these definitions, see PI Specification [57] and Section F.2.

Table 36: Boot progress codes

Name	Progress Code
Driver eXecution Environment (DXE) Core started	<code>EFI_SOFTWARE_DXE_CORE EFI_SW_DXE_CORE_PC_ENTRY_POINT</code>
DXE Variable Block NVRAM init	<code>EFI_SOFTWARE_EFI_BOOT_SERVICE BS_PC_NVRAM_INIT</code>
DXE CPU Init Begin	<code>EFI_COMPUTING_UNIT_HOST_PROCESSOR EFI_CU_PC_INIT_BEGIN</code>
Powering on and Configuring CPU	<code>EFI_COMPUTING_UNIT_HOST_PROCESSOR EFI_CU_HP_PC_POWER_ON_INIT</code>
DXE CPU Init End	<code>EFI_COMPUTING_UNIT_HOST_PROCESSOR EFI_CU_PC_INIT_END</code>
DXE SoC Devices Init	<code>EFI_COMPUTING_UNIT_CHIPSET EFI_CHIPSET_PC_DXE_SB_DEVICES_INIT</code>

Name	Progress Code
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
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 37: 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 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 refers 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, see [5] (SPDM v1.1.0 § 211 and § 292).

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, see [5] (SPDM v1.1.0 § 306).

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 can be used by the MC to decide to disable communication with devices with unknown, altered, or outdated firmware with possible security issue.

For the measurement data exchange diagram, see [5].

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, see [5] (SPDM v1.1.0 § 95).

For secure message format, see [5] (SPDM v1.0.0 § 50).

G.5 PA-RoT and AC-RoT responsibilities

The platform can optionally integrate a PA-RoT and/or an AC-RoT component. The PA-RoT can be optionally integrated in the BMC or be an isolated component. The AC-RoT, when present, must be integrated into the SoC.

Below are examples of the responsibilities of either component. The lists provide guidance only, are of informative nature and are not exhaustive.

G.5.1 PA-RoT responsibilities

1. The PA-RoT acts as the root of trust (RoT) on the platform secure boot process. The PA-RoT authenticates any firmware it loads from flash before allowing that firmware to execute.
2. The PA-RoT can control a platform Firmware Store. Writes to this Firmware Store should be first authenticated by the PA-RoT.
3. The PA-RoT can periodically monitor the Firmware Store to detect any unauthorized changes to its content.

G.5.2 AC-RoT responsibilities

1. The AC-RoT acts both as an SoC activity monitor (with the intent of detecting any ongoing malicious activity), and as a provider of security related services to the SoC.

H Firmware update

This section 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 [58].

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 [59]. 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 [41], 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. CXL and NVMe devices may use CXL [50] and NVMe [44] 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 [41], 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 [14], but that is not recommended for components that may be updated dynamically.

CXL and NVMe devices may use CXL [50] and NVMe [44] 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 [59].