

Arm[®] Base Boot Requirements 2.0

Platform Design Document



Arm Base Boot Requirements

Copyright © 2020-2022 Arm Limited or its affiliates. All rights reserved.

Release information

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

Table 1-1 Change History

Date	Issue	Confidentiality	Change
6 Oct 2020	F	Non-Confidential	Arm BBR version 1.0
15 April 2022	G	Non-Confidential	Arm BBR version 2.0 <ul style="list-style-type: none"> • Added requirements for TRNG FW API • Added CXL 2.0 UEFI/ACPI Requirements • Removed Chapter on DeviceTree • Clarified SBBR Devicetree banning rule • Clarified secondary core boot requirements • Added back the firmware update requirements • Added a reference to ACPI for Arm MPAM • Added RAS2 to recommended ACPI tables • Clarified Media I/O Protocols requirements • Added a reference to CoreSight ACPI table • Added Time and Alarm Device methods to recommended • Clarified 16550 UART ACPI DSDT requirements • Updated reference to DEN0093 ACPI for Arm Components • Added MPAM Requirements • Added ACPI IBFT support for iSCSI • Added _DSM method for SDEI Event Signaling • Added support for SMBIOS v3.6.0 • Removed reference specifications that are not referenced • Deprecated ESBBR recipe • Added reference and guidance to Arm FFH specification for CPPC and LPI • Updated LBBR recipe requirements • Clarified CXL HMAT/SRAT/SLIT requirements • Added CXL CHBCR support • Added PCIe I/O Coherency rule • Added CXL Root device support • Added reference to the APMT support • Added additional SMBIOS requirements • Clarified SMBIOS requirements for system identification • Added support for Armv9 processor family in SMBIOS • Clarified “Conditionally Required” ACPI and UEFI requirements • Added PCIe specific requirements • Added SBBR SMBIOS Firmware Revision guidance • Moved AGDI from the required section

Arm Non-Confidential Document Licence (“Licence”)

This Licence is a legal agreement between you and Arm Limited (“**Arm**”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this Licence (“**Document**”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this Licence. By using or copying the Document you indicate that you agree to be bound by the terms of this Licence.

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

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

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

- (i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;
- (ii) manufacture and have manufactured products which have been created under the licence granted in (i) above; and
- (iii) sell, supply and distribute products which have been created under the licence granted in (i) above.

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

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

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

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

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

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

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

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

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

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

Copyright © [2020-2022] 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: LES-PRE-21585 version 4.0

1	ABOUT THIS DOCUMENT	8
1.1	References	8
1.1.1	Cross References	9
1.2	Terms and abbreviations	9
1.3	Feedback	10
2	BACKGROUND	11
3	INTRODUCTION	12
4	BBR RECIPES	13
4.1	SBBR recipe	13
4.2	EBBR recipe	13
4.3	LBBR recipe	13
4.3.1	LBBR-v1	13
4.4	BBR recipes stacks	14
5	PSCI/SMCCC REQUIREMENTS	16
5.1	SMCCC Architecture Call requirements	16
5.2	PSCI Call requirements	16
5.3	Other SMC Call Requirements	16
5.3.1	TRNG Firmware Interface	16
6	SECONDARY CORE BOOT	17
7	UEFI REQUIREMENTS	18
7.1	UEFI version	18
7.2	UEFI compliance	18
7.3	UEFI system environment and configuration	18
7.3.1	AArch64 Exception levels	18
7.3.2	System volume format	18
7.3.3	UEFI image format	18
7.3.4	GOP protocol	19
7.3.5	Address translation support	19
7.4	UEFI boot services	19
7.4.1	Memory map	19
7.4.2	UEFI loaded images	19
7.4.3	Configuration tables	19
7.5	UEFI Runtime Services	20
7.5.1	Runtime Exception level	20
7.5.2	Runtime memory map	20
7.5.3	Real-time clock	20
7.5.4	UEFI reset and shutdown	21
7.5.5	Set variable	21
7.6	Firmware Update	21
8	ACPI REQUIREMENTS	23
8.1	ACPI version	23
8.2	ACPI provided data structures	23
8.3	ACPI tables	23
8.3.1	Mandatory ACPI tables	23
8.3.2	Recommended ACPI tables	26

8.3.3	Optional ACPI tables	26
8.4	ACPI definition blocks	26
8.5	ACPI methods and objects	26
8.5.1	Global methods and objects	26
8.5.2	Device methods and objects	26
8.5.3	GPIO controllers	27
8.5.4	Generic Event Devices	27
8.5.5	Address translation support	28
8.5.6	Describing Arm components in ACPI	28
8.6	Hardware requirements imposed on the platform by ACPI	28
8.6.1	Processor Performance Control	28
8.6.2	Time and Alarm Device	28
9	SMBIOS REQUIREMENTS	29
9.1	SMBIOS version	29
9.1.1	SMBIOS requirements on UEFI	29
9.2	SMBIOS structures	29
9.2.1	Type00: BIOS Information (required)	29
9.2.2	Type01: System Information (required)	29
9.2.3	Type02: Baseboard (or Module) Information (recommended)	30
9.2.4	Type03: System Enclosure or Chassis (required)	30
9.2.5	Type04: Processor Information (required)	30
9.2.6	Type07: Cache Information (required)	31
9.2.7	Type08: Port Connector Information (recommended for platforms with physical ports)	31
9.2.8	Type09: System Slots (required for platforms with expansion slots)	31
9.2.9	Type11: OEM Strings (recommended)	31
9.2.10	Type13: BIOS Language Information (recommended)	31
9.2.11	Type14: Group Associations (recommended for platforms to describe associations between SMBIOS types)	31
9.2.12	Type16: Physical Memory Array (required)	32
9.2.13	Type17: Memory Device (required)	32
9.2.14	Type19: Memory Array Mapped Address (required)	32
9.2.15	Type32: System Boot Information (required)	32
9.2.16	Type38: IPMI Device Information (required for platforms with IPMIv1.0 BMC Host Interface)	32
9.2.17	Type39: System Power Supplies (recommended for servers)	33
9.2.18	Type41: Onboard Devices Extended Information (recommended)	33
9.2.19	Type42: Redfish Host Interface (required for platforms supporting Redfish Host Interface [16])	33
9.2.20	Type43: TPM Device (required for platforms with a TPM)	33
9.2.21	Type45: Firmware Inventory Information (recommended)	33
9.2.22	Type 46: String Property (recommended)	34
10	LBBR REQUIREMENTS	35
10.1	LBBR-v1 requirements	35
10.1.1	ACPI	35
10.1.2	SMBIOS	35
10.1.3	UEFI	35
APPENDIX A	REQUIRED UEFI BOOT SERVICES	39
APPENDIX B	REQUIRED UEFI RUNTIME SERVICES	40
APPENDIX C	REQUIRED UEFI PROTOCOLS	41

APPENDIX D	OPTIONAL AND CONDITIONALLY REQUIRED UEFI PROTOCOLS	43
APPENDIX E	RECOMMENDED AND CONDITIONALLY REQUIRED ACPI TABLES	47
APPENDIX F	RECOMMENDED AND CONDITIONALLY REQUIRED ACPI METHODS	50
APPENDIX G	CXL REQUIREMENTS	53
G.1	CXL Host Bridge	53
G.2	CXL Root Device	53
G.2.1	ACPI device object for CXL Root Device	53
G.3	NUMA	53

1 About This Document

1.1 References

This document refers to the following documents:

Reference	Doc No	Authors	Title
[1]	ACPI	UEFI.org	Advanced Configuration and Power Interface Specification. Revision 6.4
[2]	Arm DDI 0487	Arm	Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile
[3]	Arm DEN 022	Arm	Power State Coordination Interface (PSCI) Version 1.1
[4]	Arm DEN 028	Arm	SMC Calling Convention (SMCCC) Version 1.2
[5]	Arm DEN 0048	Arm	Arm Functional Fixed Hardware (FFH) Specification
[6]	Arm DEN 0049	Arm	IO Remapping Table
[7]	Arm DEN 0054	Arm	Software Delegated Exception Interface (SDEI)
[8]	Arm DEN 0065	Arm	ACPI for Arm MPAM Version 1.0
[9]	Arm DEN 0067	Arm	ACPI for CoreSight Version 1.1
[10]	Arm DEN 0093	Arm	ACPI for Arm Components
[11]	Arm DEN 0094B	Arm	Arm® Base System Architecture Version 1.0b
[12]	Arm DEN 0098	Arm	Arm® True Random Number Generator Firmware Interface 1.0
[13]	Arm DEN 0117	Arm	ACPI for CoreSight PMU Architecture
[14]	CDAT	UEFI.org	Coherent Device Attribute Table (CDAT) Specification
[15]	CXL	CXL Consortium	Compute Express Link (CXL) Specification 2.0, and published ECNs and Errata
[16]	DMTF DSP0270	DMTF	Redfish Host Interface Specification Version 1.2.0
[17]	_DSD Implementation Guide	UEFI.org	_DSD (Device Specific Data) Implementation Guide
[18]	DT	Devicetree.org	Devicetree Specification 0.3

[19]	EBBR	Arm	Embedded Base Boot Requirements 2.0.1
[20]	WEG SMBIOS	Microsoft	Firmware Windows Engineering Guide
[21]	IPMI	Dell, HP, Intel, NEC	Intelligent Platform Management Interface 2.0, Revision 1.1 (October 2013)
[22]	OCP OSF Checklist	OCP	OCP Open System Firmware Checklist v1.0
[23]	PCI FW	PCI SIG	PCI Firmware Specification Revision 3.3, and published ECNs and Errata, e.g., TPH ECN
[24]	SMBIOS	DMTF	System Management BIOS (SMBIOS) Reference Specification Version 3.6.0
[25]	UEFI	UEFI.org	Unified Extensible Firmware Interface Specification. Version 2.9

1.1.1 Cross References

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

Examples:

- ACPI § 5.6.5 - Reference to the ACPI specification [1] section 5.6.6
- UEFI § 6.1 - Reference to the UEFI specification [25] section 6.1

1.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
A64	The 64-bit Arm instruction set used in AArch64 state. All A64 instructions are 32 bits.
AArch64 state	The Arm 64-bit Execution state that uses 64-bit general-purpose registers, and a 64-bit Program Counter (PC), Stack Pointer (SP), and Exception Link Registers (ELR). AArch64 Execution state provides a single instruction set, A64.
ACPI	Advanced Configuration and Power Interface
BMC	Baseboard Management Controller
DT	DeviceTree
EFI Loaded Image	An executable image to be run under the UEFI environment, and which uses boot time services.
EL0	The lowest Exception level. The Exception level that is used to execute user applications, in Non-secure state.
EL1	Privileged Exception level. The Exception level that is used to execute operating systems, in Non-secure state.

EL2	Hypervisor Exception level. The Exception level that is used to execute hypervisor code. EL2 is always in Non-secure state.
EL3	Secure monitor Exception level. The Exception level that is used to execute Secure monitor code, which handles the transitions between Non-secure and Secure states. EL3 is always in Secure state.
HDM	Host-managed Device Memory, such as CXL attached memory.
OEM	Original Equipment Manufacturer. In this document, the final device manufacturer.
PSCI	Power State Coordination Interface
SiP	Silicon Partner. In this document, the silicon manufacturer.
SMBIOS	System Management BIOS
SMCCC	SMC Calling Convention
UEFI	Unified Extensible Firmware Interface.
UEFI Boot Services	Functionality that is provided to UEFI Loaded Images during the UEFI boot process.
UEFI Runtime Services	Functionality that is provided to an operating system after the ExitBootServices() call.

1.3 Feedback

Arm welcomes feedback on its documentation.

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

- The title Arm Base Boot Requirements.
- The document ID and version (DEN0044G 2.0).
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

2 Background

Arm processors are used in a wide variety of System on Chip products in many diverse markets. The constraints on products in these markets are inevitably very different. This means that it is impossible to produce a single product that meets the needs of the various markets.

The Arm architecture profiles, Application, Real-time, and Microcontroller, segment Arm solutions and align with the functional requirements of different target markets. The differences between products that are targeted at different profiles are substantial. These differences reflect the diverse functional requirements of the market segments.

However, even within an architectural profile, the wide-ranging use of a product means that there are frequent requests for features to be removed to save silicon area. This is relevant for products that are targeted at cost-sensitive markets. In these markets, the cost of customizing software to accommodate the loss of a feature is small, compared to the overall cost saving of removing the feature itself.

In other markets, like those which require an open platform with complex software, the cost of software development to support the different variants of a hardware feature outweighs the savings that are gained from removing the variation. In addition, third parties often perform software development. The uncertainty about whether new features are widely deployed can be a substantial obstacle to the adoption of those features.

The Arm Application profile must balance these two competing business pressures. This profile offers a wide range of features, like Advanced SIMD, floating-point support, and TrustZone system security technology, to tackle an increasing range of problems. The Arm Application profile also provides the flexibility to reduce silicon space, by removing hardware features in cost-sensitive implementations.

Arm processors are built into a large variety of systems. Aspects of this system functionality are crucial to the fundamental function of system software.

Variability in boot models and certain key aspects of the system has a direct impact on the cost of software system development and the associated quality risks.

The Base Boot Requirements (BBR) specification is part of the Arm strategy of addressing this variability.

3 Introduction

This document specifies the Base Boot Requirements (BBR) for Boot and Runtime Services, based on Arm 64-bit architecture, that system software, for example operating systems and hypervisors, can rely on.

The primary goal of this document is to ensure sufficient standard system architecture to enable a suitably built single OS image to run on all hardware that is compliant with this specification and the Arm Base System Architecture specification [11]. A driver-based model for advanced platform capabilities beyond basic system configuration and boot is required. However, this model is beyond the scope of this document. Fully discoverable and describable peripherals aid the implementation of this type of a driver model.

This document identifies the Arm and industry standard firmware interfaces applicable to the Arm 64-bit architecture. They include the PSCI, SMCCC, UEFI, ACPI, SMBIOS, and DT interfaces. Requirements that are based on these interfaces are specified. In addition, various recipes are created to accommodate the various operating systems and hypervisors.

Arm does not require compliance to this specification. Arm anticipates that Cloud Service Providers (CSPs), OEMs, ODMs, and software providers will require compliance to maximize out of box software compatibility and reliability.

This document is structured as following:

Section 4 BBR recipes

Provides recipes to accommodate the various operating systems and hypervisors. This includes, but is not limited to, SBBR, EBBR, and LBBR.

Note: SBBR, EBBR and LBBR are the names of the BBR recipes. The details of the recipes are specified in Section 4. SBBR and EBBR recipes reflect the requirements specified in the SBBR (the predecessor to this document) and EBBR specifications, respectively. The recipes are designed to accommodate the various operating systems and hypervisors, regardless of the market segments.

Section 5 PSCI/SMCCC requirements

Section 6 Secondary Core Boot requirements

Section 7 UEFI requirements

Section 8 ACPI requirements

Section 9 SMBIOS requirements

Section 10 LBBR requirements

Appendixes

Provide summaries of required and recommended UEFI and ACPI interfaces.

Implementations that are consistent with Base Boot Requirements can include other features that are not included in the definition of BBR. However, software that is written for a specific version of BBR must run, unaltered, on implementations that include this type of extra functionality.

4 BBR Recipes

This section provides recipes to accommodate various operating systems and hypervisors.

These recipes define the Boot and Runtime Services for a physical system, including services that are required for virtualization. The recipes do not define a standardized abstract virtual machine view for a Guest operating system.

Note: Servers that implement SBBR or LBBR recipes can be compliant with Open Compute Project (OCP) Open System Firmware by following the [OCP OSF Checklist requirements](#) [22].

4.1 SBBR recipe

Systems using SBBR recipe must meet the requirements that are specified in section 5 (PSCI/SMCCC), section 6 (Secondary Core Boot), section 7 (UEFI), section 8 (ACPI), and section 9 (SMBIOS).

Currently, Windows Server, Windows 10, Red Hat Enterprise Linux (RHEL), VMware ESXi, Amazon Linux, and Oracle Linux require SBBR recipe. Other operating systems, for example SUSE Linux Enterprise Server (SLES), CentOS, Ubuntu, NetBSD, and FreeBSD can also support SBBR. This list of operating systems and hypervisors is subject to change.

Note: The reference implementation that is provided at tianocore.org is called EDK2. However, SBBR compliance does not require EDK2 implementation.

4.2 EBBR recipe

Systems using EBBR recipe must meet the requirements that are specified in section 5 (PSCI/SMCCC) and in the EBBR specification v2.0.1[19].

Note: The EBBR specification defines a reduced UEFI environment. The underlying implementation of EBBR specification is typically U-Boot. However, EBBR does not preclude EDK2 implementation.

Note: Currently, Fedora, Debian, openSUSE, SLES, and Ubuntu support EBBR. This list of operating systems and hypervisors is subject to change.

4.3 LBBR recipe

LBBR is a recipe for LinuxBoot based systems. [LinuxBoot](#) is an alternative firmware stack that uses the Linux kernel as the Normal world firmware component. LinuxBoot is not a standard, but it can be supported on platforms that provide open source firmware and implement the requirements listed in this section.

LinuxBoot recipe is defined in a phased approach, using multiple versions. LBBR-v1 is the first of those versions, which is defined as a practical set of requirements that map to the Arm server implementations in 2021-2022. The goal is to continue evolving the LBBR recipe to reduce the dependency on underlying UEFI firmware implementations, and at the same time, improve the standard firmware interfaces published by LinuxBoot to the final operating system.

Note: LinuxBoot is used by some hyperscale datacenters that are compliant with the Open Compute Project (OCP) [Open System Firmware \(OSF\) Checklist](#) [22].

4.3.1 LBBR-v1

Systems using LBBR-v1 recipe must meet the requirements in section 5 (PSCI/SMCCC), section 6 (Secondary Core Boot), section 7 (UEFI), section 8 (ACPI), section 9 (SMBIOS), with the exceptions that are specified in Section 10.1 (LBBR-v1).

The list of operating systems and hypervisors that can support LBBR-v1 is as follows and subject to change:

- CentOS

- Fedora
- Debian
- Ubuntu

4.4 BBR recipes stacks

The following diagram shows the relationships among SBBR, EBBR, and LBBR recipes:

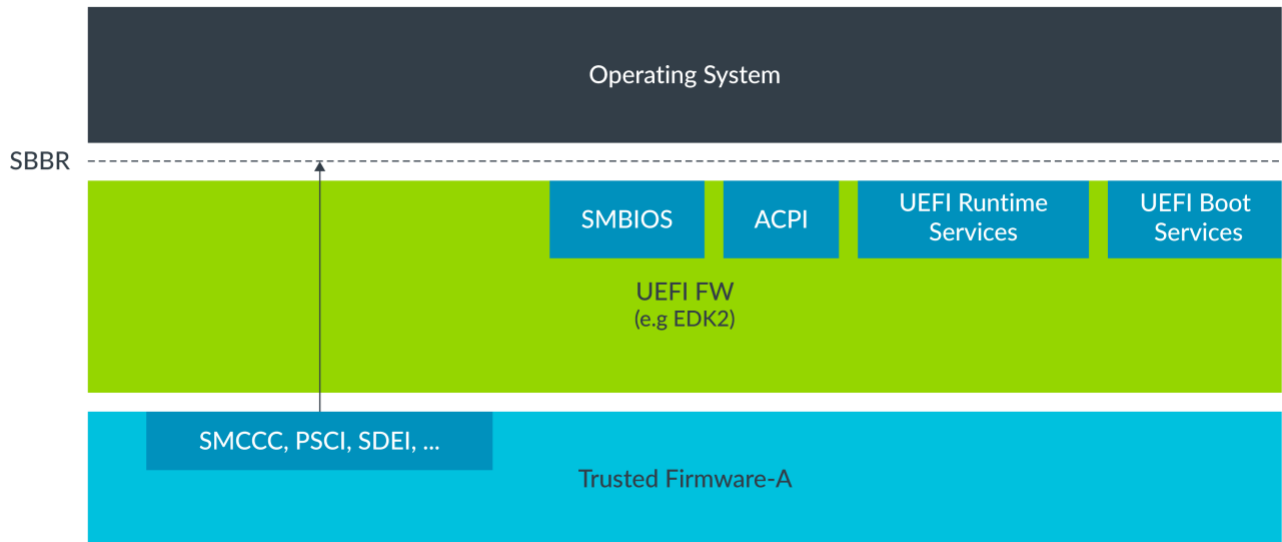


Figure 4-1 SBBR recipe firmware stack example

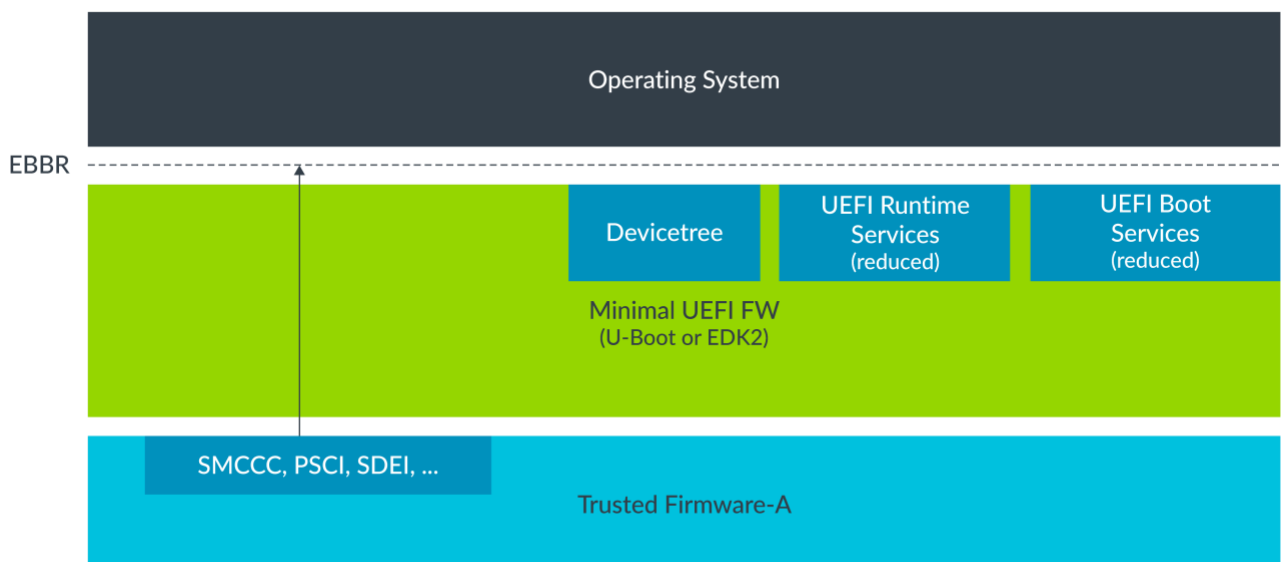


Figure 4-2 EBBR recipe firmware stack example

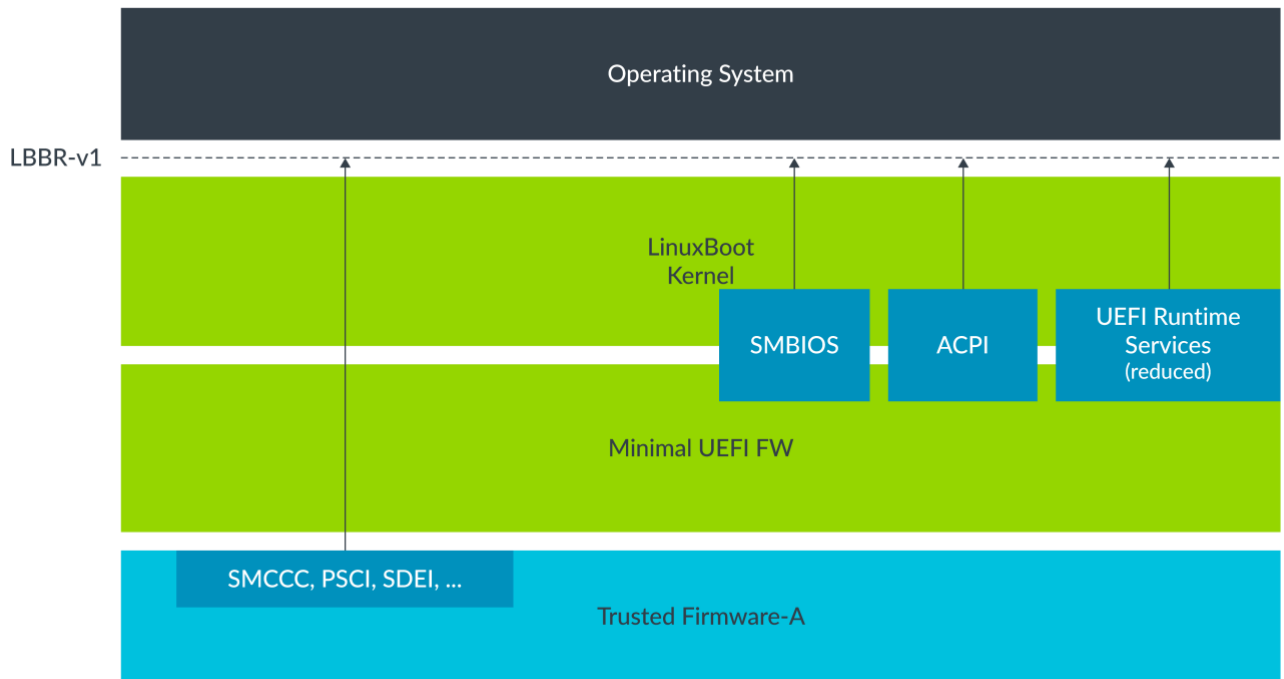


Figure 4-3 LBBR recipe firmware stack example

5 PSCI/SMCCC Requirements

The system must expose PSCI [3] and SMCCC [4] interfaces to the operating system or hypervisor.

Note: Trusted Firmware-A (TF-A) provides a reference implementation of Secure world software that is executing at EL3. TF-A implements various Arm interface standards including PSCI and SMCCC.

5.1 SMCCC Architecture Call requirements

Arm Architecture Call	SMCCC §	Requirement
SMCCC_VERSION	7.2	Required for platforms that support SMCCC v1.1 or newer.
SMCCC_ARCH_FEATURES	7.3	Required for platforms that support SMCCC v1.1 or newer.
SMCCC_ARCH_SOC_ID	7.4	Required for platforms that support SMCCC v1.2 or newer. On such platforms, both SoC_ID types 0 (SoC version) and 1 (SoC revision) must be implemented.
SMCCC_ARCH_WORKAROUND_1	7.5	Recommended for platforms that support SMCCC v1.1 or newer and contain at least one PE that is affected by CVE-2017-5715.
SMCCC_ARCH_WORKAROUND_2	7.6	Recommended for platforms that support SMCCC v1.1 newer and contain at least one PE that is affected by CVE-2018-3639.

5.2 PSCI Call requirements

The system must meet PSCI 1.1 compliance requirements as defined in [PSCI § 6.9]:

- All mandatory functions must be implemented.
- Optional functions are recommended to be implemented.

5.3 Other SMC Call Requirements

This section describes additional requirements or conditional requirements for other SMC calls.

5.3.1 TRNG Firmware Interface

If the system implements an entropy source in privileged software (secure world), and would like to expose it to the normal world (e.g. Operating System) via an SMCCC interface, it must do so using the Arm True Random Number Generator (TRNG) Firmware Interface defined in [12].

6 Secondary Core Boot

Platforms providing EL3 must implement the Power State Coordination Interface (PSCI) [3]:

PSCI 1.1 Published April 2017

This interface will be the main method for booting secondary cores, implementing CPU idling, and providing reset and shutdown runtime services.

ACPI tables need to reflect:

- FADT should indicate the presence of PSCI.
- MADT GICC structures must provide valid MPIDR entries.

Where CPU idling low power states are provided, the DSDT must provide `_LPI` objects.

Functional Fixed Hardware (FFH) can be used to specify low power state entry methods and low power state residency and usage statistics. See Arm FFH Specification [5] for more details.

Secondary cores can be brought online for specific tasks by calling `CPU_ON()` into PSCI firmware. A secondary core brought online by firmware services before OS load must be put offline again by calling `CPU_OFF()` into PSCI firmware to force the cores into OS compatible state before handing off control to OS. If UEFI is used, this should happen before completing the `EFI_EVENT_GROUP_READY_TO_BOOT` event. After boot, OSPM can call `CPU_ON()` into PSCI firmware to start a chosen secondary core.

7 UEFI Requirements

7.1 UEFI version

This document references the following specification and versions:

UEFI 2.9 [25] Published March 2021, includes the AArch64 bindings

7.2 UEFI compliance

Any UEFI-compliant system must follow the requirements that are laid out in section 2.6 of the UEFI specification. Systems that are compliant with this section must always provide the UEFI services and protocols that are listed in Appendix A, Appendix B, and Appendix C of this document.

UEFI Protocols that are optional are listed in Appendix D. Not every platform that is compliant with this specification provides all of these protocols. This is because many protocols reference optional platform features. For example, a platform does not have to implement the UEFI networking protocols unless the platform supports booting the OS from the network. If the platform supports OS booting from the network, then the appropriate network boot technology protocols (PXE, HTTP(s), and/or iSCSI) must be implemented.

7.3 UEFI system environment and configuration

7.3.1 AArch64 Exception levels

The resident AArch64 UEFI boot-time environment is specified to use the highest 64-bit Non-secure privilege level available. This level is either EL1 or EL2, depending on whether virtualization is used or supported.

Resident UEFI firmware might target a specific Exception level. In contrast, UEFI loaded images, like third-party drivers and boot applications, must not contain any built-in assumptions of the Exception level to be loaded at boot time. This is because these UEFI loaded images can be loaded into EL1 or EL2.

7.3.1.1 UEFI Boot at EL2

Systems must boot UEFI at EL2, to allow for the installation of a hypervisor or a virtualization-aware operating system.

7.3.1.2 UEFI Boot at EL1

Booting of UEFI at EL1 is only permitted within a Guest operating system environment, to allow the subsequent booting of a UEFI-compliant operating system. In this instance, the UEFI boot-time environment can be provided as a virtualized service by the hypervisor, and not part of the host firmware.

7.3.2 System volume format

The system firmware must support the disk partitioning schemes that are required by the UEFI specification [UEFI §2.6.2][UEFI §13.3.1].

7.3.3 UEFI image format

UEFI allows the extension of platform firmware, by loading UEFI driver and UEFI application images [UEFI § 2.1]

7.3.3.1 UEFI drivers

A device may provide a container for one or more UEFI drivers [UEFI § 2.1.4] that are used for the device initialization. If a platform supports the inclusion or addition of such a device, at least one of the UEFI drivers must be in the A64 binary format.

7.3.3.2 UEFI applications

A UEFI application [UEFI § 2.1.2] must be in the A64 binary format to be used for the systems that comply with this specification.

7.3.3.3 PE/COFF image

The SectionAlignment and FileAlignment fields, as defined in [Microsoft PE Format](#), must contain the value of at least 4KiB. Higher values are also permitted, for example for DXE_RUNTIME_DRIVER modules, to meet the 64KiB granular memory type requirements that are imposed by the UEFI specification.

Modules that may execute in place, for example SEC, PEI_CORE or PEIM type UEFI/PI modules, are exempt from this requirement. For these modules, any power-of-2 value of 32 bytes or higher is permitted, if the section alignment and file alignment are equal.

PE/COFF images whose section alignment is at least 4KiB should not contain any sections that have both the IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE attributes set.

7.3.4 GOP protocol

For systems with graphics video hardware, [EFI_GRAPHICS_OUTPUT_PROTOCOL](#) (UEFI §12.9) is recommended to be implemented with the frame buffer of the graphics adapters directly accessible (for example [EFI_GRAPHICS_PIXEL_FORMAT](#) is not PixelBltOnly). The GOP FrameBufferBase must be reported as a CPU physical address, not as a bus address (like a PCI(e) bus address).

7.3.5 Address translation support

If a platform includes PCI bus support, then the [EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL](#) (UEFI §14.2) and the [EFI_PCI_IO_PROTOCOL](#) (UEFI §14.4) must be implemented. The implementation of the [EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL](#) must provide the correct Address Translation Offset field to translate between the host and bus addresses. [EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_CONFIGURATION](#) must report resources produced by the PCI(e) root bridge, not resources consumed by its register maps. In the cases where there are unpopulated PCIe slots behind the root bridge, [EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_CONFIGURATION](#) must report valid resources assigned (for example, for hot plug), or report no resources assigned.

7.4 UEFI boot services

7.4.1 Memory map

The UEFI environment must provide a system memory map, which must include all appropriate devices and memories that are required for booting and system configuration.

All RAM defined by the UEFI memory map must be identity-mapped. This means that virtual addresses must have equal physical addresses.

The default RAM allocated attribute must be [EFI_MEMORY_WB](#).

7.4.2 UEFI loaded images

UEFI loaded images for AArch64 must be in 64-bit PE/COFF format and must contain only A64 code.

7.4.3 Configuration tables

A UEFI system that complies with this specification must provide the following tables through the EFI Configuration Table:

- [EFI_ACPI_20_TABLE_GUID](#)
 - The ACPI tables must be at version ACPI 6.4 or later with a HW-Reduced ACPI model. See section 8.
- [SMBIOS3_TABLE_GUID](#)

- This table defines the 64-bit entry point for SMBIOS table.
- The SMBIOS tables must conform to version 3.6.0 or later of the SMBIOS Specification. See section 9.

A UEFI system that complies with this specification can provide the following tables through the EFI Configuration Table, with the following conditions:

- **EFI_DTB_TABLE_GUID**
 - This optional table defines the platform Devicetree in Flattened Devicetree Blob (DTB) format [18]
 - Firmware can use a minimal Devicetree to provide necessary information to the OS bootloader, as long as this does NOT include any platform hardware description. The platform hardware description is provided using ACPI abstractions.
 - For example, the following are acceptable properties that can be included in this optional devicetree:


```
/chosen/linux,initrd-start
/chosen/linux,initrd-end
```

7.5 UEFI Runtime Services

UEFI Runtime Services exist after `ExitBootServices()` is called. UEFI Runtime Services provide a limited set of persistent services to the platform operating system or hypervisor.

The Runtime Services that are listed in Appendix B must be provided.

7.5.1 Runtime Exception level

UEFI enables runtime services to be supported at either EL1 or EL2, with appropriate virtual address mappings. When called, subsequent runtime service calls must be from the same Exception level.

7.5.2 Runtime memory map

Before calling `ExitBootServices()`, the final call to `GetMemoryMap()` returns a description of the entire UEFI memory map. This description includes the persistent Runtime Services mappings.

After the call to `ExitBootServices()`, the Runtime Services page mappings can be relocated in virtual address space by calling `SetVirtualAddressMap()`. This call allows the Runtime Services to assign virtual addresses that are compatible with the incoming operating system memory map.

A UEFI runtime environment that is compliant with this specification must not be written with any assumption of an identity mapping between virtual and physical memory maps.

UEFI operates with a 4KiB page size. With Runtime Services, these pages are mapped into the operating system address space.

To allow operating systems to use 64KiB page mappings, the UEFI specification constrains all mapped 4KiB memory pages to have identical page attributes within the same physical 64K page.

7.5.3 Real-time clock

The Real-time Clock must be accessible through the UEFI runtime firmware, and the following services must be provided:

- `GetTime()`
- `SetTime()`

It is permissible for `SetTime()` to return an error on systems where the Real-time Clock cannot be set by this call.

7.5.4 UEFI reset and shutdown

The UEFI Runtime service `ResetSystem()` must implement the following `ResetType` values, for purposes of power management and system control:

- `EfiResetCold`
- `EfiResetShutdown`
 - `EfiResetShutdown` must not reboot the system.

If firmware updates are supported through the Runtime Service of `UpdateCapsule()`, then `ResetSystem()` might need to support the following command: `EfiResetWarm`.

These Runtime Services must be implemented by calling into PSCI. The following table maps the UEFI and PSCI reset calls:

EFI <code>ResetSystem()</code> <code>ResetType</code>	PSCI Reset call
<code>EfiResetShutdown</code>	<code>SYSTEM_OFF</code>
<code>EfiResetCold</code>	<code>SYSTEM_RESET</code>
<code>EfiResetWarm</code>	<code>SYSTEM_RESET2</code> , with <code>reset_type = 0x0</code> (<code>SYSTEM_WARM_RESET</code>)
<code>EfiResetPlatformSpecific</code> , with a platform-specific <code>ResetData GUID</code> .	The exact mapping of the UEFI runtime call to the PSCI call is IMPLEMENTATION DEFINED.

Note: When Runtime Services and PSCI co-exist, operating systems may call either interface to reset the system. Calling the `UpdateCapsule()` service requires the use of the UEFI Runtime service for reset.

7.5.5 Set variable

Non-volatile UEFI variables must persist across reset, and emulated variables in RAM are not permitted.

The UEFI Runtime Services must be able to update the variables directly without the aid of the operating system.

Note: UEFI variables normally require dedicated storage that is not directly accessible from the operating system.

7.6 Firmware Update

BBR platforms are required to implement either an in-band or an out-of-band firmware update mechanism.

If the firmware update is performed in-band (firmware on the application processor updates itself), then the firmware must implement `EFI_UPDATE_CAPSULE` (UEFI § 8.5.3). Firmware must accept updates in the "Firmware Management Protocol Data Capsule Structure" format as described in "Delivering Capsules Containing Updates to Firmware Management Protocol" (UEFI § 23.3). Firmware must also provide an ESRT (UEFI § 23.4) that describes every firmware image that is updated in-band.

If the firmware update is performed out-of-band (for example, by an independent Baseboard Management Controller (BMC)), then the platform is not required to implement `EFI_UPDATE_CAPSULE`.

`EFI_UPDATE_CAPSULE` is only required before `ExitBootServices()` is called.

The `EFI_UPDATE_CAPSULE` implementation is expected to be suitable for use by generic firmware update services like fwupd and Windows Update. Both fwupd and Windows Update read the ESRT table to determine what firmware can be updated and use an EFI helper application to call `EFI_UPDATE_CAPSULE` before `ExitBootServices()` is called.

Note: If an OS uses UEFI capsule services, the OS must clean the data cache by VA on each `ScatterGatherList` element that is passed to the `UpdateCapsule()` before issuing the call. The cache clean operation used must be the strongest available. For example, Data cache clean to Point of Persistence (DC CVAP) introduced in v8.2 or Cache Clean to Point of Deep Persistence introduced in v8.5. This is required only when `UpdateCapsule()` is called after `ExitBootServices()`. This requirement is clarified in UEFI specification (UEFI § 8.5.3).

8 ACPI Requirements

8.1 ACPI version

This document references the following specification and versions:

ACPI 6.4[16] [1] Published January 2021

ACPI is used to describe the hardware resources that are installed, and to handle aspects of runtime system configuration, event notification, and power management.

The ACPI-compliant OS must be able to use ACPI to configure the platform hardware and provide basic operations. The ACPI tables are passed, through UEFI, into the OS to drive the Operating System-directed Power Management (OSPM).

This section defines mandatory and optional ACPI features, and a few excluded features.

8.2 ACPI provided data structures

All platforms that comply with this specification must:

- Conform to the ACPI specification[16], version 6.4 or later. Legacy tables and methods are not supported.
- Implement the HW-Reduced ACPI model. See ACPI § 3.11.1 and 4.1.
- Not support legacy ACPI Fixed Hardware interfaces
- Provide either Interrupt-signaled Events (see ACPI § 5.6.9) or GPIO-signaled Events (see ACPI § 5.6.5) for the conveyance of runtime event notifications, from the system firmware to the Operating System Power Management (OSPM).

8.3 ACPI tables

ACPI tables are essentially data structures. The OSPM of the operating system receives a pointer to the Root System Description Pointer (RSDP) from the boot loader. The OSPM then uses the information in the RSDP to determine the addresses of all other ACPI tables. Platform designers can decide whether the ACPI tables are stored in ROM or in flash memory.

All platforms that comply with this specification:

- Must ensure that the structure of all tables is consistent with the ACPI 6.4 or later specification. Legacy tables are not supported.
- Must ensure that the pointer to the RSDP is passed through UEFI to the OSPM as described by UEFI
- Must use 64-bit addresses within all address fields in ACPI tables
 - This restriction ensures a long-term future for the ACPI tables. Versions before ACPI 5.0 allowed 32-bit address fields.

8.3.1 Mandatory ACPI tables

The following tables are mandatory for all compliant systems.

8.3.1.1 RSDP

- Root System Description Pointer (RSDP), ACPI § 5.2.5.
 - Within the RSDP, the RsdtdAddress field must be null (zero) and the XsdtAddress must be a valid, non-null, 64-bit value.

8.3.1.2 XSDT

- Extended System Description Table (XSDT), ACPI § 5.2.8.
 - The RSDP must contain a pointer to this table.
 - This table contains pointers to all other ACPI tables that the OSPM will use.

8.3.1.3 FADT

- Fixed ACPI Description Table (FADT), ACPI § 5.2.9
 - The ACPI signature for this table is actually FACP. The name FADT is used for historical reasons.
 - This table must have the HW_REDUCED_ACPI flag set to comply with the HW-Reduced ACPI model. Many other fields must be set to null when this flag is set.
 - The recommendation for the Preferred PM Profile is that an appropriate profile as defined by the ACPI specification (ACPI § 5.2.9.1) is selected.
 - For servers, the recommendation for the Preferred PM Profile is that one of the server profiles is selected.
 - The ARM_BOOT_ARCH flags describe the presence of PSCI. See ACPI § 5.2.9.4.

8.3.1.4 DSDT and SSDT

- Differentiated System Description Table (DSDT), ACPI § 5.2.11.1.
 - This table provides the essential configuration information that is needed to boot the platform.
- Secondary System Description Table (SSDT), ACPI § 5.2.11.2.
 - This table is optional. One or more of SSDT can be used to provide definition blocks if necessary.

8.3.1.5 MADT

- Multiple APIC Description Table (MADT), ACPI § 5.2.12.
 - This table describes the GIC interrupt controllers, their version, and their configuration.
 - For systems without PSCI, this table provides the Parked Address for secondary CPU initialization.
- The strong recommendation for the entry order of GICC structures is that it reflects affinity in resource sharing, typically caches, in the system. That is, processors that share resources should be close together in the ordering. For example, consider an SMT system with the following properties:
 - Comprised of two sockets, in which each socket has a large cache that is shared by all logical processors in the socket.
 - Each socket contains two processor clusters, and within each cluster there is cache that is shared by all logical processors in the cluster.
 - Each physical processor contains two logical processors or hardware threads, which share physical processor resources.
- The recommended indexing in the order of GICC structures for this system should increase in hardware thread, then cluster, and then socket ordering. In other words, it should be:
 - Socket 0, Cluster 0, Thread 0 – GICC
 - Socket 0, Cluster 0, Thread 1 – GICC
 - Socket 0, Cluster 1, Thread 0 – GICC
 - Socket 0, Cluster 1, Thread 1 – GICC
 - Socket 1, Cluster 0, Thread 0 – GICC
 - Socket 1, Cluster 0, Thread 1 – GICC

- Socket 1, Cluster 1, Thread 0 – GICC
- Socket 1, Cluster 1, Thread 1 – GICC

8.3.1.6 GTDT

- Generic Timer Descriptor Table (GTD), ACPI § 5.2.24.
 - This table describes the Arm Generic Timer block and the Arm Generic Watchdog.

8.3.1.7 DBG2

- Debug Port Table 2 (DBG2). See <http://uefi.org/acpi>
 - This table provides a standard debug port.
 - This table can be used to describe the UART as specified by BSA §3.12. When doing so, the serial sub-type values must follow the requirements in the table below (see section 8.3.1.8).
 - **Note:** If OS debug through serial port is wanted, the system must make the UART instance that is specified in DBG2 to be exclusively available for use of the debugger. How the system enables this is IMPLEMENTATION DEFINED.

8.3.1.8 SPCR

- Serial Port Console Redirection (SPCR). See <http://uefi.org/acpi>
 - This table provides the essential configuration information that is needed for headless operations, like a kernel shell or console.
 - This table defines a serial port type, location, and interrupts.
 - **Note:** This table can be used to describe the UART as specified by BSA §3.12. When doing so, the Interface Type values must follow the requirements in the table below.
- This specification requires revision 2 or later of the SPCR table. Revisions before 2 are not supported.
- The SPCR must be populated with correct ACPI GSIV interrupt routing information for the UART device.
- The SPCR console device must be included in the DSDT, as described in ACPI for Arm Components §2.3 [10] and in the table below:

UART hardware	SPCR and DBG2 serial port type	DSDT _HID
16550 compliant	0x01 (for existing OSes) 0x12 (for future OSes)	IMPLEMENTATION DEFINED
Prime cell UART (PL011)	0x03	ARMH0011
Arm Generic UART	0x0E	ARMH0011 (for existing OSes) ARMHB000 (for future OSes)

8.3.1.9 MCFG

- PCI Memory-mapped Configuration Space (MCFG). PCI FW [23] § 4.1.2
 - This table described the PCIe ECAM base address
 - This table is required if PCIe is supported.

8.3.1.10 PPTT

- Processor Properties Topology Table (PPTT), ACPI § 5.2.29.

- This table describes the topological structure of processors that are controlled by the OSPM and their shared resources.

8.3.2 Recommended ACPI tables

ACPI tables that are recommended are listed in Appendix E .

Not every platform that is compliant with this specification provides all of these tables. This is because many tables reference optional platform features. For example, a platform does not have to implement NUMA for memory. If it does, it must provide the SRAT and SLIT that describe the NUMA topology to ACPI. In addition, HMAT can also be used to describe the heterogeneous memory attributes.

Note: There are exceptional cases. For instance, if a platform supports CXL-attached memory, it might be required to support the SRAT and HMAT tables. See Appendix G for more details on rules related to CXL.

8.3.3 Optional ACPI tables

All other tables that are defined in the ACPI specification can be used as needed for AArch64 platforms, but only if they comply with syntax and semantics of the specification.

8.4 ACPI definition blocks

Within the DSDT or SSDT, tables that are used to describe the platform, devices are defined by ACPI definition blocks (see ACPI § 5.2.11). Each of these definition blocks describes one or more devices that cannot be enumerated by the OSPM at boot time without more information. For example, processors must be described by definition blocks, but PCI devices are enumerated by a defined protocol.

8.5 ACPI methods and objects

A DSDT or SSDT definition block contains definitions of objects and methods which can be invoked. These definitions can provide global information, but most of them provide information that is specific to a single device. Objects and methods can also be predefined, either by the ACPI specification or as needed by a platform designer.

All objects and methods must conform to the definitions in ACPI version 6.4 or later. Legacy definitions are not supported.

ACPI methods that are recommended are listed in Appendix F. Not every platform that is compliant with this specification provides all of these methods. This is because many methods reference optional platform features. For example, a platform does not have to implement the SDEI _DSM method, unless the platform supports SDEI-based event signaling.

8.5.1 Global methods and objects

Platforms must define processors as devices under the _SB (System Bus) namespace. See ACPI § 5.3.1

Platforms must not define processors using the global _PR (Processors) namespace. See ACPI § 5.3.1

Platforms that comply with this specification can provide the following predefined global methods:

- `_SST`: System Status Indicator. This method reports on the current overall state of the system status indicator, if and only if a platform provides a user-visible status like an LED.
 - See ACPI § 9.2.1

8.5.2 Device methods and objects

For each device definition in the platform DSDT or SSDT tables, platforms must provide the following predefined methods or objects, in accordance with their definitions in version 6.4 or later of the ACPI specification:

- **_ADR:** Address on the parent bus of the device. Either this object or the **_HID** must be provided. This object is essential for PCI, for example.
 - See ACPI § 6.1.1
- **_CCA:** Cache Coherency Attribute. This object provides information about whether a device has to manage cache coherency and about hardware support. This object is mandatory for all devices that are not cache-coherent, and recommended for all devices. This object is only relevant for devices that can access CPU-visible memory, like devices that are DMA capable.
 - See ACPI § 6.2.17
- **_CRS:** Current Resource Settings. This method provides essential information to describe resources, like registers and their locations, that are provided by the device.
 - See ACPI § 6.2.2.
 - **Note:** The **_PRS** (Possible Resource Settings) and **_SRS** (Set Resource Settings) are not supported.
- **_HID:** Hardware ID. This object provides the Plug and Play Identifier or the ACPI ID for the device. Either this object or the **_ADR** must be provided.
 - See ACPI § 6.1.5.
- **_STA:** Status. This method identifies whether the device is on, off, or removed.
 - See ACPI § 6.3.7 and 7.2.4.
- **_UID:** Unique persistent ID. This object provides a unique value that is persistent across boots, and can uniquely identify the device with either a common **_HID** or **_CID**. The object is used, for example, to identify a PCI root bridge, if there are multiple PCI root bridges in the system.
 - See ACPI § 6.1.12.

Note: A **_HID** object must be used to describe any device that is enumerated by OSPM. OSPM only enumerates a device when no bus enumerator can detect the ID. For example, devices on an ISA bus are enumerated by OSPM. Use the **_ADR** object to describe devices that are enumerated by bus enumerators other than OSPM.

8.5.3 GPIO controllers

The HW-Reduced ACPI model has specific requirements for GPIO controllers and devices. If a platform supports GPIO-signaled ACPI events, it must provide the following methods:

- **_AEI:** ACPI Event Interrupts. This object defines which GPIO interrupts are to be handled as ACPI events.
 - See ACPI § 5.6.5.2.
- **_EVT:** Event method for GPIO-signaled interrupts. For event numbers that are less than 255, the **_Exx** or **_Lxx** methods can be used instead.
 - See ACPI § 5.6.5.3 and 5.6.4.1.

8.5.4 Generic Event Devices

The HW-Reduced ACPI model has specific requirements for Generic Event Devices. Platforms that support interrupt-signaled ACPI events must provide the Generic Event Devices with the following methods:

- **_CRS:** Current Resource Setting. This object designates those interrupts that shall be handled by OSPM as ACPI events.
 - See ACPI § 5.6.9.2
- **_EVT:** Event method for interrupt-signaled interrupts.
 - See ACPI § 5.6.9.3.

8.5.5 Address translation support

PCIe-compliant devices are recommended, eliminating the need to support legacy I/O port space. However, if the platform supports legacy I/O port space, it must report the host (CPU) to the PCI I/O bus address space translations using resource descriptors of type DWordIO, QWordIO or ExtendedIO. The TranslationType must be set to TypeStatic. This is because of existing OS behavior.

8.5.6 Describing Arm components in ACPI

Certain components that are implemented or licensed by Arm have special ACPI properties and IDs defined in ACPI for Arm Components [10]. If a platform includes any of the Arm components that are listed in ACPI for Arm Components, it must follow the guidance of that specification for implementing the ACPI objects and methods for such components.

8.6 Hardware requirements imposed on the platform by ACPI

The term HW-Reduced does not imply anything about functionality. HW-Reduced simply means that the hardware specification is not implemented. See Chapter 4 of the ACPI specification. All functionality is still supported through equivalent software-defined interfaces.

Instead, the complexity of the OSPM in supporting ACPI is reduced. For example, many requirements from versions earlier than version 5.0 can be ignored. However, this model does impose some requirements on the hardware that is provided by the platform. In particular, either interrupt-signalled events (ACPI § 5.6.9) or GPIO-signalled events (ACPI § 5.6.5) must be used to generate interrupts that are functionally equivalent to General Purpose Events (GPEs). See ACPI § 5.6.4.

Platforms that comply with this specification must provide the following platform events:

- For the ACPI Platform Error Interface (APEI):
 - One event for non-fatal error signalling (ACPI § 18.3.2.7.2)
 - Software Delegated Exception(SDE) [7] or one NMI-equivalent signal for use in fatal errors
 - See ACPI § 18.
- At least one wake signal, which is routed through a platform event.

Note: for systems that do not support Sx states except S5 soft off, this can be just the power button.

8.6.1 Processor Performance Control

If OSPM-directed processor performance control is supported, then it must be exposed using Collaborative Processor Performance Control (CPPC).

The use of Platform Communications Channel (PCC) is highly recommended for processor performance management. See ACPI § 14. Using PCC address space allows to support process performance management in operating systems which do not support flexible address space for CPPC Registers. See Bit[14] of Table 6-200 Platform-Wide _OSC Capabilities, ACPI § 6.2.11.2.

Note: Provisioning a Platform Interrupt is recommended if PCC is used. See Platform Communications Channel Global Flags, ACPI § 14.1.1.

Arm cores which implement the Activity Monitor Unit (AMU) can use Functional Fixed Hardware (FFH) to monitor the performance of a logical processor. See Arm FFH Specification [5] for more details.

8.6.2 Time and Alarm Device

If the ACPI Time and Alarm Device is implemented (see ACPI § 9.18), it must operate on the same real-time clock that is exposed by the UEFI Runtime Services.

9 SMBIOS Requirements

The System Management BIOS (SMBIOS) [24] that is published by the DMTF is an important firmware component for computer systems. SMBIOS provides basic hardware and firmware configuration information through table-driven data structures. Although it is not required for operating system booting or core kernel functions, SMBIOS is widely used for platform management, scripting, and deployment applications.

9.1 SMBIOS version

This document references the following specification and versions:

SMBIOS 3.6.0 Published (as a WIP) December 2021.

Legacy SMBIOS tables and formats are not supported.

9.1.1 SMBIOS requirements on UEFI

- UEFI uses SMBIOS3_TABLE_GUID to identify the SMBIOS table.
- UEFI uses the EfiRuntimeServicesData type for the system memory region containing the SMBIOS table.
- UEFI must not use the EfiBootServicesData type for the SMBIOS data region, as the region could be reclaimed by a UEFI-compliant operating system after UEFI ExitBootServices() is called.

9.2 SMBIOS structures

SMBIOS implementations vary by system design and form factor. For a BBR-compliant system, the following SMBIOS structures are required or recommended. For required data within these structures, please refer to Table 4 and Annex A of the SMBIOS specification.

Unless specified otherwise in the SMBIOS specification, all “human readable” strings in SMBIOS structures must be UTF-8 encoded, using US-ASCII characters.

For additional guidance and requirements on SMBIOS tables reporting for telemetry and servicing, refer to [20].

9.2.1 Type00: BIOS Information (required)

- Vendor
- BIOS Version
 - This field must match the BIOS firmware version string displayed in firmware user interfaces or referenced in documentation
- BIOS Release Date
- BIOS ROM Size
- System BIOS Major Release
- System BIOS Minor Release
 - System BIOS Major and Minor Release fields must not have values equal to 0FFh. The numeric values should correspond to the major and minor portions of the BIOS Version string
- Embedded Controller Firmware Major Release
- Embedded Controller Firmware Minor Release
- Extended BIOS ROM Size

9.2.2 Type01: System Information (required)

- Manufacturer
 - This field must identify the system manufacturer company name
- Product Name
 - This field must identify the company specific model of the system
- Version

- Serial Number
 - This field must identify the individual system serial number
- UUID
 - This field must provide a unique value for every individual system
- SKU Number
 - This field must provide a system configuration identification
- Family
 - This field must identify the company specific sub-brand name of the system

9.2.3 Type02: Baseboard (or Module) Information (recommended)

- Manufacturer
- Product
- Version
- Serial Number
- Asset Tag
- Location in Chassis
- Board Type

9.2.4 Type03: System Enclosure or Chassis (required)

- Manufacturer
- Type
- Version
- Serial Number
- Asset Tag Number
 - The value of this field is recommended to be from a user-settable string
- Height
- SKU Number
- Enclosure Type

9.2.5 Type04: Processor Information (required)

- Socket Designation
- Processor Type
- Processor Family
 - This field must provide a human readable description of the processor product line.
- Processor Manufacturer
 - This field must provide a human readable description of the processor manufacturer.
- Processor ID
 - For systems that support SMBIOS 3.4.0 or newer and support SMCCC_ARCH_SOC_ID call, this field must implement the SoC Id value, as specified by SMBIOS specification. Otherwise, this field must implement the MIDR_EL1 value as specified by SMBIOS specification.
- Processor Version
 - This field must provide a human readable description of the processor part number.
- Max Speed
- Status
- Core Count
- Core Enabled
- Thread Count
- Processor Family 2
- Core Count 2
- Core Enabled 2
- Thread Count 2

Exactly one Type04 structure must be provided for every socket in the system. For example, N Type04 structures, in a one-to-one mapping with each physical socket, out of a socket count of N.

- A physical socket is defined as a discrete SoC, or equivalent physical chip package (implementing a chip-to-chip extension of cache coherency, and typically participating within the same Inner Shareable domain as defined in [2]).

9.2.6 Type07: Cache Information (required)

- Socket Designation
- Cache Configuration
- Maximum Cache Size
- Installed Size
- Cache Speed
- Maximum Cache Size 2
- Installed Cache Size 2

9.2.7 Type08: Port Connector Information (recommended for platforms with physical ports)

- Internal Reference Designator
- Internal Connector Type
- External Reference Designator
- External Connector Type
- Port Type

9.2.8 Type09: System Slots (required for platforms with expansion slots)

- Slot Designation
- Slot Type
- Slot Data Bus Width
- Current Usage
- Slot ID
- Slot Characteristics 1
- Slot Characteristics 2
- Segment Group Number
- Bus Number
- Device Function Number
- Peer grouping count
- Peer groups
 - Slots that contain multiple devices are recommended to report all devices using peer groups

9.2.9 Type11: OEM Strings (recommended)

- Count
 - This field is needed only if vendor specific strings are described

9.2.10 Type13: BIOS Language Information (recommended)

- Installable Languages
- Flags
- Current Language

9.2.11 Type14: Group Associations (recommended for platforms to describe associations between SMBIOS types)

- Group Name

- Item Type
- Item Handle

This SMBIOS type can be used to describe association between SMBIOS types, such as associating memory device (Type 17) to the processors they are attached to (Type 4).

9.2.12 Type16: Physical Memory Array (required)

- Location
- Use
- Maximum Capacity
- Number of Memory Devices
- Extended Maximum Capacity

9.2.13 Type17: Memory Device (required)

- Total Width
- Data Width
- Size
- Device Locator
- Memory Type
- Type Detail
- Speed
- Manufacturer
- Serial Number
- Asset Tag
- Part Number
- Extended Size
- Extended Speed

In addition, the following fields are required if NVDIMM is supported:

- Memory Technology
- Memory Operating Mode Capability
- Non-volatile Size
- Volatile Size
- Cache Size
- Logical Size

9.2.14 Type19: Memory Array Mapped Address (required)

- Starting Address
- Ending Address
- Extended Starting Address
- Extended Ending Address

9.2.15 Type32: System Boot Information (required)

- Boot Status

9.2.16 Type38: IPMI Device Information (required for platforms with IPMIv1.0 BMC Host Interface)

- IPMI Specification Revision
- I2C Target Address
- Base Address
- Base Address Modifier
- Interrupt Info

- Interrupt Number

Note: The ACPI SPMI Table replaces this Type in IPMI v1.5 and v2.0 [21].

9.2.17 Type39: System Power Supplies (recommended for servers)

- Location
- Device Name
- Manufacturer
- Serial Number
- Asset Tag Number
- Model Part Number
- Revision Level
- Max Power Capacity

Note: This applies only to power supplies that have a management/communication path to UEFI or BMC.

9.2.18 Type41: Onboard Devices Extended Information (recommended)

- Reference Designation
- Device Type
- Device Type Instance
- Segment Group Number
- Bus Number
- Device Function Number

9.2.19 Type42: Redfish Host Interface (required for platforms supporting Redfish Host Interface [16])

- Interface Type
- Interface Specific Data
 - Device Type must be 04h (USB Network Interface v2) or 05h (PCI/PCIe Network Interface v2).
- Protocol Records

9.2.20 Type43: TPM Device (required for platforms with a TPM)

- Vendor ID
- Major Spec Version
- Minor Spec Version
- Firmware Version 1
- Firmware Version 2
- Description
- Characteristics

9.2.21 Type45: Firmware Inventory Information (recommended)

- Firmware Component Name
- Firmware ID
- Firmware ID Format
- Release Date
- Manufacturer
- Characteristics
- State
- Number of Associated Components
- Associated Components Handles

This SMBIOS type can be used to describe firmware components in the system, such as TF-A, SCP, NIC firmware, and so on. Systems that provide firmware inventory with a BMC (using Redfish for instance) and publish this SMBIOS type must correlate the data in this SMBIOS type with the data provided by the BMC.

Note: Firmware inventory provided in this SMBIOS type is a static snapshot taken at the end of the system boot. It does not reflect dynamic changes to firmware components versions due to out-of-band updates or hot plug of devices. Systems that support dynamic FW updates should not solely rely on this SMBIOS table for reporting FW inventory.

The entries in this table are platform specific, and not standardized. The following list provides a guidance on sample use cases for an SMBIOS Type 45 implementation:

- System Controllers
 - Could be multiple entries, describing different System Controllers in the system, depending on whether their FW is reported independently or not. Examples are SCP, SatMC or MCP, CXL Fabric Manager, and so on.
- Secure Firmware
 - Could be a single entry that covers all TF-A, or multiple entries for different Secure World / TF-A components, such as BL1, BL2, BL31, Secure Partition Manager, Secure OS, or different BL32 secure partitions.
- Platform Firmware
 - This is the BL33 platform boot loader firmware, such as "UEFI firmware".
- Option Devices
 - Could be an entry per PCIe option device (such as a NIC) that presents a firmware version via UEFI Firmware Management Protocol (FMP).

9.2.22 Type 46: String Property (recommended)

- String Property ID
- String Property Value
- Parent handle

Systems that provide hardware inventory through a BMC (using Redfish for instance) can use this SMBIOS type to report the UEFI Device Path strings of various system components and correlate them to other SMBIOS structures.

10 LBBR Requirements

10.1 LBBR-v1 requirements

The system firmware for LBBR-v1 recipe must meet the requirements specified in this section. With this approach, LinuxBoot kernel exposes the required interfaces (UEFI Runtime, ACPI, and SMBIOS) to be used by the final runtime Linux kernel. The final Linux kernel is loaded using kexec, since the UEFI boot services are not available for this recipe. There is an implied assumption that ExitBootServices() has already occurred prior to the kexec call to load the final kernel. There is also an implied assumption that hardware initialization, such as PCIe link training, is completed prior to the kexec to the final kernel.

Note: The reference implementation of the LinuxBoot reduced UEFI firmware that is provided at tianocore.org is called EDK2. However, LBBR-v1 compliance does not require EDK2 implementation.

10.1.1 ACPI

The system firmware must meet all the requirements in section 8 (ACPI), with no exceptions.

10.1.2 SMBIOS

The system firmware must meet all the requirements in section 9 (SMBIOS), with no exceptions.

10.1.3 UEFI

The system firmware must meet all the requirements in section 7 (UEFI) with exceptions, as outlined in the table below:

BBR §	LBBR-v1 requirements exception	Notes
7.1 - UEFI version	No	All requirements apply
7.2 0 UEFI compliance	Yes	LBBR-v1 defines several exceptions for the requirements in: <ul style="list-style-type: none"> Section 2.6 of the UEFI specification Appendix A, Appendix B, and Appendix C of this document For details of the exceptions, refer to 10.1.3.1.
7.3.1 AArch64 Exception Levels	No	All requirements apply
7.3.2 – System volume format	Yes	Not applicable for LBBR-v1
7.3.3.1 - UEFI image format – UEFI drivers	Yes	Not applicable for LBBR-v1
7.3.3.2 – UEFI image format – UEFI applications	Yes	Not applicable to LBBR-v1
7.3.3.3 – UEFI image format – PE/COFF image	Yes	Requirements apply only to the LinuxBoot kernel image, using EFI Stub.

7.3.4 – GOP Protocol	Yes	These requirements are optional for LBBR-v1 compliant systems that wish to pass the UEFI video frame buffer to the final Operating System.
7.3.5 Address translation support	Yes	Not applicable for LBBR-v1
7.4.1 UEFI boot services - memory Map	No	All requirements apply
7.4.2 – UEFI Loaded Images	Yes	Requirements apply only to the LinuxBoot kernel image, using EFI Stub.
7.4.3 – Configuration tables	No	All requirements apply
7.5.1 – Runtime Exception Level	No	All requirements apply
7.5.2 – Runtime memory map	No	All requirements apply
7.5.3 - Real-time clock	Yes	All UEFI time runtime services are optional for LBBR-v1.
7.5.4 - UEFI reset and shutdown	Yes	All UEFI reset and shutdown services are optional for LBBR-v1. If implemented, the requirements of mapping the UEFI runtime services to PSCI apply.
7.5.5 – Set variable	Yes	All UEFI variables runtime services are optional for LBBR-v1.
7.6 – Firmware Update	Yes	In-band firmware update using UEFI capsule runtime services are optional for LBBR-v1
Appendix A – Required UEFI Boot Services	Yes	None of the UEFI Boot Services are required to be implemented for LBBR-v1.
Appendix B - Required UEFI Runtime Services	Yes	For details of UEFI runtime services requirements and exceptions, see 10.1.3.2.
Appendix C – Required UEFI Protocols	Yes	None of the UEFI Protocols are applicable to LBBR-v1.
Appendix D – Optional UEFI Protocols	Yes	None of the UEFI Protocols are applicable to LBBR-v1.

10.1.3.1 Deviation from UEFI Specification requirements

LBBR-v1 compliant systems must conform to a subset of the UEFI Specification [25] as listed in this section. Normally, UEFI compliance would require full compliance with all items listed in UEFI § 2.6. However, the LBBR target market has a reduced set of requirements, and so some UEFI features are omitted as unnecessary.

UEFI §	LBBR-v1 requirements exception	Notes
2.6.1 - Required Elements	Yes	<code>EFI_SYSTEM_TABLE</code> and <code>EFI_RUNTIME_SERVICES</code> are required for LBBR-v1 systems. All UEFI System

		<p>Table function pointers (including the UEFI Runtime Services) must be available and cannot be NULL pointers. Some UEFI runtime services can return <code>EFI_UNSUPPORTED</code>, as detailed in the requirements in section 10.1.3.2 below.</p> <p>The firmware-defined console (ConIn/ConOut/StdErr) pointers in the System Table are recommended to be implemented, even if they are stubs, to avoid potential runtime access errors.</p> <p>All other elements, including <code>EFI_BOOT_SERVICES</code>, are either not required or are not applicable to LBBR-v1.</p>
2.6.2 - Platform Specific Elements	Yes	None of the platform specific UEFI elements, including all UEFI protocols, are required or applicable to LBBR-v1.
3.3 - Globally Defined Variables	Yes	<p>None of the Globally Defined UEFI Variables are required for LBBR-v1 systems.</p> <p>Note: Some system operations (such as Boot Order control, and UEFI Secure Boot database management) may be limited if the globally defined UEFI variables are missing. These operations may require platform specific methods to provide the equivalent functionality.</p>

10.1.3.2 UEFI Runtime Services requirements

UEFI runtime services exist after the call to `ExitBootServices()` and are designed to provide a limited set of persistent services to the operating system or hypervisor. Functions contained in `EFI_RUNTIME_SERVICES` are normally available during both boot services and runtime services. However, for LBBR-v1, `EFI_RUNTIME_SERVICES` functions are not applicable before `ExitBootServices` is called(). They are required to be implemented and available after `ExitBootServices()`, and cannot be NULL pointers. In addition, it is possible for some of these functions to not be available after `ExitBootServices()` is called. In that case, the firmware shall provide the `EFI_RT_PROPERTIES_TABLE` to indicate which functions are available during runtime services. Functions that are not available during runtime services shall return `EFI_UNSUPPORTED`.

The table below details which `EFI_RUNTIME_SERVICES` are required to be implemented during boot services and runtime services.

EFI_RUNTIME_SERVICES function	Before ExitBootServices()	After ExitBootServices()
GetTime	N/A	Optional
SetTime	N/A	Optional
GetWakeupTime	N/A	Optional
SetWakeupTime	N/A	Optional
SetVirtualAddressMap	N/A	Required
ConvertPointer	N/A	Required

GetVariable	N/A	Optional
GetNextVariableName	N/A	Optional
SetVariable	N/A	Optional
GetNextHighMonotonicCount	N/A	Optional
ResetSystem	N/A	Optional
UpdateCapsule	N/A	Optional
QueryCapsuleCapabilities	N/A	Optional
QueryVariableInfo	N/A	Optional

Appendix A Required UEFI Boot Services

Service	UEFI §
EFI_RAISE_TPL	7.1
EFI_RESTORE_TPL	7.1
EFI_ALLOCATE_PAGES	7.2
EFI_FREE_PAGES	7.2
EFI_GET_MEMORY_MAP	7.2
EFI_ALLOCATE_POOL	7.2
EFI_FREE_POOL	7.2
EFI_CREATE_EVENT	7.1
EFI_SET_TIMER	7.1
EFI_WAIT_FOR_EVENT	7.1
EFI_SIGNAL_EVENT	7.1
EFI_CLOSE_EVENT	7.1
EFI_INSTALL_PROTOCOL_INTERFACE	7.3
EFI_REINSTALL_PROTOCOL_INTERFACE	7.3
EFI_UNINSTALL_PROTOCOL_INTERFACE	7.3
EFI_HANDLE_PROTOCOL	7.3
EFI_REGISTER_PROTOCOL_NOTIFY	7.3
EFI_LOCATE_HANDLE	7.3
EFI_LOCATE_PROTOCOL	7.3
EFI_LOCATE_DEVICE_PATH	7.3
EFI_INSTALL_CONFIGURATION_TABLE	7.5
EFI_LOAD_IMAGE	7.4

EFI_START_IMAGE	7.4
EFI_EXIT	7.4
EFI_UNLOAD_IMAGE	7.4
EFI_EXIT_BOOT_SERVICES	7.4
EFI_GET_NEXT_MONOTONIC_COUNT	7.5
EFI_STALL	7.5
EFI_SET_WATCHDOG_TIMER	7.5
EFI_CONNECT_CONTROLLER	7.3
EFI_DISCONNECT_CONTROLLER	7.3
EFI_OPEN_PROTOCOL	7.3
EFI_CLOSE_PROTOCOL	7.3
EFI_OPEN_PROTOCOL_INFORMATION	7.3
EFI_PROTOCOLS_PER_HANDLE	7.3
EFI_LOCATE_HANDLE_BUFFER	7.3
EFI_LOCATE_PROTOCOL	7.3
EFI_INSTALL_MULTIPLE_PROTOCOL_INTERFACES	7.3
EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES	7.3
EFI_CALCULATE_CRC32	7.5
EFI_COPY_MEM	7.5
EFI_SET_MEM	7.5
EFI_CREATE_EVENT_EX	7.1

Appendix B Required UEFI Runtime Services

Service	UEFI §
EFI_GET_TIME	8.3
EFI_SET_TIME	8.3
EFI_GET_WAKEUP_TIME	8.3
EFI_SET_WAKEUP_TIME	8.3
EFI_SET_VIRTUAL_ADDRESS_MAP	8.4
EFI_CONVERT_POINTER	8.4
EFI_GET_VARIABLE	8.2
EFI_GET_NEXT_VARIABLE_NAME	8.2
EFI_SET_VARIABLE	8.2
EFI_GET_NEXT_HIGH_MONO_COUNT	8.5
EFI_RESET_SYSTEM	8.5
EFI_UPDATE_CAPSULE	8.5
EFI_QUERY_CAPSULE_CAPABILITIES	8.5
EFI_QUERY_VARIABLE_INFO	8.2

Note: `EFI_GET_WAKEUP_TIME` and `EFI_SET_WAKEUP_TIME` must be implemented, but might simply return `EFI_UNSUPPORTED`. `EFI_UPDATE_CAPSULE` and `EFI_QUERY_CAPSULE_CAPABILITIES` must be implemented, but might simply return `EFI_UNSUPPORTED`.

UEFI Configuration Table Entries

Configuration Table
EFI_ACPI_20_TABLE_GUID
SMBIOS3_TABLE_GUID

Appendix C Required UEFI Protocols

Core UEFI Protocols

Service	UEFI §
EFI_LOADED_IMAGE_PROTOCOL	9.1
EFI_LOADED_IMAGE_DEVICE_PATH_PROTOCOL	9.2
EFI_DECOMPRESS_PROTOCOL	19.5
EFI_DEVICE_PATH_PROTOCOL	10.2
EFI_DEVICE_PATH_UTILITIES_PROTOCOL	10.5

Media I/O Protocols

Service	UEFI §
EFI_LOAD_FILE_PROTOCOL	13.1
EFI_LOAD_FILE2_PROTOCOL	13.2
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL	13.4
EFI_FILE_PROTOCOL	13.5

The Load File protocol is used to obtain files from arbitrary devices that are primarily boot options. This includes, for example, booting from network devices. The Load File 2 protocol is used to obtain files from arbitrary devices that are not boot options. The Media I/O protocols are required to be present only when booting from or accessing media devices that require the corresponding protocol(s).

Console Protocols

Service	UEFI §
EFI_SIMPLE_TEXT_INPUT_PROTOCOL	12.3
EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL	12.2
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL	12.4

Driver Configuration Protocols

Service	UEFI §
EFI_HII_DATABASE_PROTOCOL	34.8
EFI_HII_STRING_PROTOCOL	34.3
EFI_HII_CONFIG_ROUTING_PROTOCOL	35.4

EFI_HII_CONFIG_ACCESS_PROTOCOL	35.5
--------------------------------	------

Random Number Generator Protocol

Service	UEFI §
EFI_RNG_PROTOCOL	37.5

The `EFI_RNG_PROTOCOL` is used by operating systems for entropy generation capabilities early during OS boot. The protocol is recommended to support returning at least 256 bits of full entropy in a single call, from a source with security strength of at least 256 bits. For instance, it is possible that this protocol implementation is backed by the Arm TRNG Firmware Interface [12].

Appendix D Optional and Conditionally Required UEFI Protocols

This section lists optional UEFI protocols. These may be conditionally required for some platforms, depending on the platform features.

Basic Networking Support

Service	UEFI §
EFI_SIMPLE_NETWORK_PROTOCOL	24.1
EFI_MANAGED_NETWORK_PROTOCOL	25.1
EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL	25.1

Networking services are optional on platforms that do not support networking.

Network Boot Protocols

Service	UEFI §
EFI_PXE_BASE_CODE_PROTOCOL	24.3
EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL	24.4
EFI_MTFTP4_PROTOCOL	30.3
EFI_MTFTP6_PROTOCOL	30.4

Ipv4 Network Support

Service	UEFI §
EFI_ARP_PROTOCOL	29.1
EFI_ARP_SERVICE_BINDING_PROTOCOL	29.1
EFI_DHCP4_SERVICE_BINDING_PROTOCOL	29.2
EFI_DHCP4_PROTOCOL	29.2
EFI_TCP4_PROTOCOL	28.1.2
EFI_TCP4_SERVICE_BINDING_PROTOCOL	28.1.1
EFI_IP4_SERVICE_BINDING_PROTOCOL	28.3.1
EFI_IP4_CONFIG2_PROTOCOL	28.5
EFI_UDP4_PROTOCOL	30.1.2
EFI_UDP4_SERVICE_BINDING_PROTOCOL	30.1.1

Networking services are optional on platforms that do not support networking.

Ipv6 Networking Support

Service	UEFI §
EFI_DHCP6_PROTOCOL	29.3.2
EFI_DHCP6_SERVICE_BINDING_PROTOCOL	29.3.1
EFI_TCP6_PROTOCOL	28.2.2
EFI_TCP6_SERVICE_BINDING_PROTOCOL	28.2.1
EFI_IP6_SERVICE_BINDING_PROTOCOL	28.6.1
EFI_IP6_CONFIG_PROTOCOL	28.7
EFI_UDP6_PROTOCOL	30.2.2
EFI_UDP6_SERVICE_BINDING_PROTOCOL	30.2.1

Networking services are optional on platforms that do not support networking.

VLAN Protocols

Service	UEFI §
EFI_VLAN_CONFIG_PROTOCOL	27.1

iSCSI Protocols

Service	UEFI §
EFI_ISCSI_INITIATOR_NAME_PROTOCOL	16.2

Support for iSCSI is only required on machines that lack persistent storage, like an HDD. This configuration is intended for thin clients and compute-only nodes.

REST Protocols

Service	UEFI §
EFI_REST_EX_PROTOCOL	29.7.2.2
EFI_REST_EX_SERVICE_BINDING_PROTOCOL	29.7.2.1
EFI_REDFISH_DISCOVER_PROTOCOL	31.1.4
EFI_REST_JSON_STRUCTURE_PROTOCOL	29.7.3.2

Support for REST protocol is optional on machines that support RESTful communication (for example, Redfish Host Interface to a BMC).

HTTP Network Protocols

Service	UEFI §
EFI_HTTP_SERVICE_BINDING_PROTOCOL	29.6.1
EFI_HTTP_PROTOCOL	29.6.2
EFI_HTTP_UTILITIES_PROTOCOL	29.6.3
EFI_HTTP_BOOT_CALLBACK_PROTOCOL	24.7.6
EFI_DNS4_SERVICE_BINDING_PROTOCOL	29.4
EFI_DNS4_PROTOCOL	29.4
EFI_DNS6_SERVICE_BINDING_PROTOCOL	29.5.1
EFI_DNS6_PROTOCOL	29.5.2
EFI_TLS_SERVICE_BINDING_PROTOCOL	28.10.1
EFI_TLS_PROTOCOL	28.10.2
EFI_TLS_CONFIGURATION_PROTOCOL	28.10.3

Networking services are optional on platforms that do not support networking.

Firmware Update

Service	UEFI §
EFI_FIRMWARE_MANAGEMENT_PROTOCOL	23.1
EFI_SYSTEM_RESOURCE_TABLE_GUID (ESRT) (UEFI Configuration Table)	23.4

Firmware Management Protocol is used for device firmware updates if the device UEFI driver supports it.

The ESRT configuration table is used for firmware updates if the system implements UEFI capsule services.

CXL UEFI Protocols

Service	§
EFI_ADAPTER_INFORMATION_PROTOCOL, with EFI_ADAPTER_INFO_CDAT_TYPE_GUID type	UEFI §11.12.6
CXL CDAT Table Access Data Object Exchange (DOE) method	CXL §8.1.11

Platforms supporting CXL devices with coherent memory are required to support extracting Coherent Device Attribute Table (CDAT) [14] structures from the devices, using one of the methods described above. The following requirements also apply:

- Platforms supporting CXL 1.1 devices must follow the rules outlined in CXL § 9.11, including the System Firmware view (CXL § 9.11.2), the System Firmware enumeration (CXL § 9.11.4), and the CXL device discovery flow (CXL § 9.11.5)

- Platforms supporting CXL 2.0 devices must follow the rules outlined in CXL 2.0 Enumeration (CXL § 9.12) and CXL OS Firmware Interface Extensions (CXL § 9.14)

See Appendix G for more details on rules related to CXL.

Appendix E Recommended and Conditionally Required ACPI Tables

This section lists recommended ACPI tables. These may be conditionally required for some platforms, depending on the platform features.

I/O Topology

IORT describes the SMMU or ITS that is required if such capabilities are supported. Components behind an SMMU that are not enumerable behind a PCIe root complex must be described as IORT nodes in the IORT table.

ACPI Signature	Full Name	ACPI §
IORT	IO Remapping Table	https://uefi.org/acpi

If an I/O subsystem supports I/O coherency, this must be represented by configuring the Cache Coherent Attribute to “coherent” in the I/O Remapping Table entries of corresponding components. If an I/O subsystem does not support I/O coherency, this must be represented by configuring the Cache Coherent Attribute to “non-coherent” in the I/O Remapping Table entries of corresponding components. See “I/O coherency: memory types and attributes for PCI Express” in [11] and “CCA: Cache Coherent Attribute” in [6].

Platform Error Interfaces

The following tables are required to support ACPI Platform Error Interfaces (APEI), which convey error information to the operating system.

ACPI Signature	Full Name	ACPI §
BERT	Boot Error Record Table	18.3.1
EINJ	Error Injection Table	18.6.1
ERST	Error Record Serialization Table	18.5
HEST	Hardware Error Source Table	18.3.2
SDEI	Software Delegated Exception Interface Table	https://uefi.org/acpi
AEST	Arm Error Source Table	https://uefi.org/acpi
RAS2	ACPI RAS2 Feature Table	ACPI code-first ECR

NUMA

The following tables describe topology and resources that are required by NUMA systems.

ACPI Signature	Full Name	ACPI §
SLIT	System Locality Information Table	5.2.17
SRAT	System Resource Affinity Table	5.2.16

HMAT	Heterogeneous Memory Attribute Table	5.2.27
------	--------------------------------------	--------

Platform Communications Channel (PCC)

PCCT provides the interface to communicate to an on-platform controller.

ACPI Signature	Full Name	ACPI §
PCCT	Platform Communications Channel Table	14

Platform Debug Trigger

PDDT describes one or more PCC subspace identifiers that can be used to trigger/notify the platform specific debug facilities to capture non-architectural system state. This is intended as a standard mechanism for the OSPM to notify the platform of a fatal crash (e.g. kernel panic or bug check).

ACPI Signature	Full Name	ACPI §
PDDT	Platform Debug Trigger Table	5.2.28

NVDIMM Firmware Interface

NFIT describes NVDIMM that is required if NVDIMM is supported.

ACPI Signature	Full Name	ACPI §
NFIT	NVDIMM Firmware Interface Table	5.2.25

Graphics Resource Table

BGRT describes system graphics resources when a video frame buffer is present.

ACPI Signature	Full Name	ACPI §
BGRT	Boot Graphics Resource Table	5.2.22

IPMI

SPMI describes the processor-relative, translated, fixed resources of an IPMI system interface at system boot time.

ACPI Signature	Full Name	ACPI §
SPMI	Server Platform Management Interface Table, if and only if IPMI has been implemented.	https://uefi.org/acpi

CXL

The following tables are required to support CXL Host Bridges in the operating system. The operating system may use this information to configure CXL.cache and CXL.mem devices.

ACPI Signature	Full Name	§
----------------	-----------	---

CEDT	CXL Early Discovery Table	CXL § 9.14.1
------	---------------------------	--------------

See Appendix G for more details on rules related to CXL.

MPAM

If a platform supports MPAM, then the boot firmware must publish an MPAM table in ACPI to support discovery and configuration of the system's MPAM topology from the operating system. The MPAM table must be created according to the guidelines outlined in the ACPI for MPAM Specification [8].

On systems with MSCs that can generate MSIs, the firmware must describe the MSCs as Named Components in the ACPI IORT table. Each Named Component node that describes an MSI-capable MSC must include an ID-mapping that describes the mapping between the MSC and the GIC ITS unit group that handles the MSI generated by the MSC.

Note: The above steps ensure that an MPAM-aware operating system can correctly discover, configure and use the MPAM capabilities of the system.

ACPI Signature	Full Name	ACPI §
MPAM	Arm Memory Partitioning and Monitoring	https://uefi.org/acpi

iSCSI

Support for iSCSI is only required on machines that lack persistent storage, like an HDD. This configuration is intended for thin clients and compute-only nodes.

ACPI Signature	Full Name	§
IBFT	iSCSI Boot Firmware Table	https://uefi.org/acpi

Debug and Performance

ACPI Signature	Full Name	§
APMT	Arm Performance Monitoring Unit Table	https://uefi.org/acpi

Diagnostic Dump Device

The AGDI Table [10] describes a generic diagnostic dump device that can be used to request the OS to perform a crash dump.

ACPI Signature	Full Name	§
AGDI	Arm Generic Diagnostic Dump and Reset Device Interface table	https://uefi.org/acpi and [10]

Appendix F Recommended and Conditionally Required ACPI Methods

This section lists recommended and conditionally required ACPI methods.

CPU performance control

For CPU performance and control, two mutually exclusive methods are defined. The newer `_CPC` method is used in conjunction with the PCCT.

Method	Full Name	ACPI §
<code>_CPC</code>	Continuous Performance Control (replaces <code>_PCT</code> and <code>_PSS</code>)	8.4.7.1
<code>_PSS</code>	Performance Supported States (Superseded by <code>_CPC</code>)	8.4.6.2

CPU and system idle control

For CPU and system idle management, the following method has been introduced in ACPI 6.0.

Method	Full Name	ACPI §
<code>_LPI</code>	Low Power Idle States	8.4.4

NUMA

The following methods describe topology and resources that are required by NUMA systems.

ACPI Signature	Full Name	ACPI §
<code>_PXM</code>	Proximity	6.2.14
<code>_SLI</code>	System Locality Information	6.2.15
<code>_HMA</code>	Heterogeneous Memory Attributes	6.2.18

IPMI

The following methods describe the interface type and revision of an IPMI system interface at system boot time.

Method	Full Name	ACPI §
<code>_IFT</code>	IPMIv2: the IPMI Interface Type, if IPMI has been implemented	5.5.2.4.4, IPMI spec [21]
<code>_SRV</code>	IPMIv2: the IPMI revision that is supported by the platform, if IPMI has been implemented	IPMI spec

Device Configuration and Control

The following methods provide configuration and control for devices.

Method	Full Name	ACPI §
_CLS	Class code (for non-PCI devices that are compatible with PCI drivers)	6.1.3
_CID	Compatible ID	6.1.2
_DSD	Device Specific Data: Provides more device properties and information. Implementations are recommended to follow the guidance of the UEFI Forum as outlined in the DSD Implementation Guide: https://github.com/UEFI/DSD-Guide [17]	6.2.5
_DSM	Device Specific Method (used to convey info to ACPI that it might not currently have a mechanism to describe, see https://lkml.org/lkml/2013/8/20/556)	9.1.1
_INI	Initialize a device	6.5.1

Resources

The following methods provide descriptions for devices.

Method	Full Name	ACPI §
_MLS	Human readable description in multiple languages. Note: this is preferred over _STR.	6.1.7
_STR	Device description (in a single language). Superseded by _MLS.	6.1.10

CXL

The following methods are required to support CXL Host Bridges in the operating system. The operating system may use this information to configure CXL.cache and CXL.mem devices.

Method	Full Name	§
_CBR	CXL Host Bridge Register Info	ACPI § 6.5.11
_OSC	CXL Operating System Capabilities	CXL § 9.14.2

See Appendix G for more details on rules related to CXL.

Performance and Debug

Method	Full Name	ACPI §
_DSD	Device Graph _DSD for Arm CoreSight – UUID 3ECBC8B6-1D0E-4FB3-8107-E627F805C6CD [9]	https://uefi.org/acpi

Time and Alarm

Method	Full Name	ACPI §
_GCP	Get the capabilities of the time and alarm device	9.18
_GRT	Get the Real time	9.18
_SRT	Set the Real time	9.18
_GWS	Get Wake status	9.18
_CWS	Clear Wake Status	9.18
_STP	Sets expired timer wake policy for the specified timer.	9.18
_STV	Sets the value in the specified timer.	9.18
_TIP	Returns the current expired timer policy setting of the specified timer	9.18
_TIV	Returns the remaining time of the specified timer	9.18

SDEI

Devices that use SDEI-based event signalling must use the _DSM method recommended in Appendix E of the SDEI specification [7], to describe the event number being used for the signalling.

Method	Full Name	§
_DSM	SDEI _DSM	SDEI § E

PCI and PCIe Express

Method	Full Name	§
_CBA	Memory-mapped Configuration Base Address	PCI FW
_DSM	_DSM definitions for PCI	PCI FW
_OSC	PCI Express capabilities	PCI FW

Note: The _CBA method must be present if the system supports hot plug of host bridges. For the _DSM method:

- If Steering Tags for cache locality are supported, then Function Index = 0Fh for Cache Locality TPH Features must be implemented, see TPH ECN[23].
- If the firmware reserves memory regions using Reserved Memory Range nodes in the IORT [6], then Function Index = 05h for preserving boot configuration must be present on the host bridge below which the devices that use the reserved memory regions reside.
- If firmware controls DPC features, then Function Index = 0Ch must be implemented.

Various PCI-specific capabilities supported by the platform must be declared in the _OSC method.

Appendix G CXL Requirements

G.1 CXL Host Bridge

For each CXL host bridge in the system, the firmware must produce a CHBS (CXL Host Bridge Structure) entry in the ACPI CEDT table to allow discovery of the CHBCR from the OS. Details of the CEDT table and the CHBS structure are available in the CXL 2.0 specification [15].

G.2 CXL Root Device

A CXL root device is the origin of a hierarchy of CXL HDM and caches. The main purpose of a CXL root device is to allow the OSPM to discover this origin and be able to enumerate CXL HDM below that origin. The root device is also used for enabling interleaving across CXL host bridges. The root device is also required for specifying global CXL-specific properties related to CXL HDM. In particular, the QoS Throttling Groups (QTG) are related to the bandwidth and latency properties described by the root device. The CXL specification defined a CXL-specific _DSM method for QTG discovery. This _DSM method must be a child of the root device that it is related to.

This section describes rules and recommendations related to the CXL root device and the associated _DSM method.

G.2.1 ACPI device object for CXL Root Device

For each CXL root complex (or an origin of a CXL hierarchy) that needs to advertise its properties to the OS, the firmware should create an ACPI CXL root device object in ACPI namespace. The root device is recognized by its HID value of "ACPI0017". The CXL root device is described in detail in the CXL specification [15].

Note: The CXL Root Device HID value is defined in this [ACPI code-first ECR](#).

For each CXL root device whose properties need to be conveyed to the OS, the firmware must include the following methods as a child of the root device:

- _DSM, function for retrieving QoS Throttling Group (QTG) IDs

G.3 NUMA

If a CXL device with HDM is present in a system, then the CDAT table presented by the device must be used to obtain NUMA properties of the HDM. Furthermore, if presented to the OS, the EFI memory map, and the HMAT, SRAT and CEDT tables must be populated with the information obtained from the CDAT tables.

If CXL-attached HDM is present in a system, and if the CDAT DSEMTS structure advertised by the device indicates that the HDM is normal memory (EfiConventionalMemory or EfiConventionalMemory with the EFI_MEMORY_SP attribute set), then it is recommended that the HMAT table is provided to describe the NUMA properties of the HDM.

It is also recommended that if the HMAT table is accompanied by an SRAT table to describe the HDM. The SRAT table is a prerequisite for the HMAT table, as specified in the ACPI specification.

If CXL HDM is present, and described in the SRAT table, the table must include a Generic Port entry in the SRAT table for CXL HDM to describe the performance/NUMA properties of the path between the PEs and the CXL gateway/bridge associated with each CXL host bridge. The Generic Port entry should indicate whether it supports all architectural features related to memory behind the port or not. The Architectural Transactions field of the Generic Port structure must be set to 1 if all memory-specific architectural features are supported. Conversely, if one or more such architectural features are not supported, the Architectural Transactions field must be set to 0.

Note: The SRAT Generic Port entry is defined in this [ACPI code-first ECR](#).

If the HDM is not described in the HMAT and SRAT tables, the system must provide a CFMWS structure in the CEDT to enable the OS to dynamically create a NUMA node for the HDM.

The SRAT table, if present, must include a Generic Initiator entry to describe CXL-attached memory initiators.

The SRAT table, if present, must include a memory affinity structure to describe the NUMA properties of the CXL HDM.