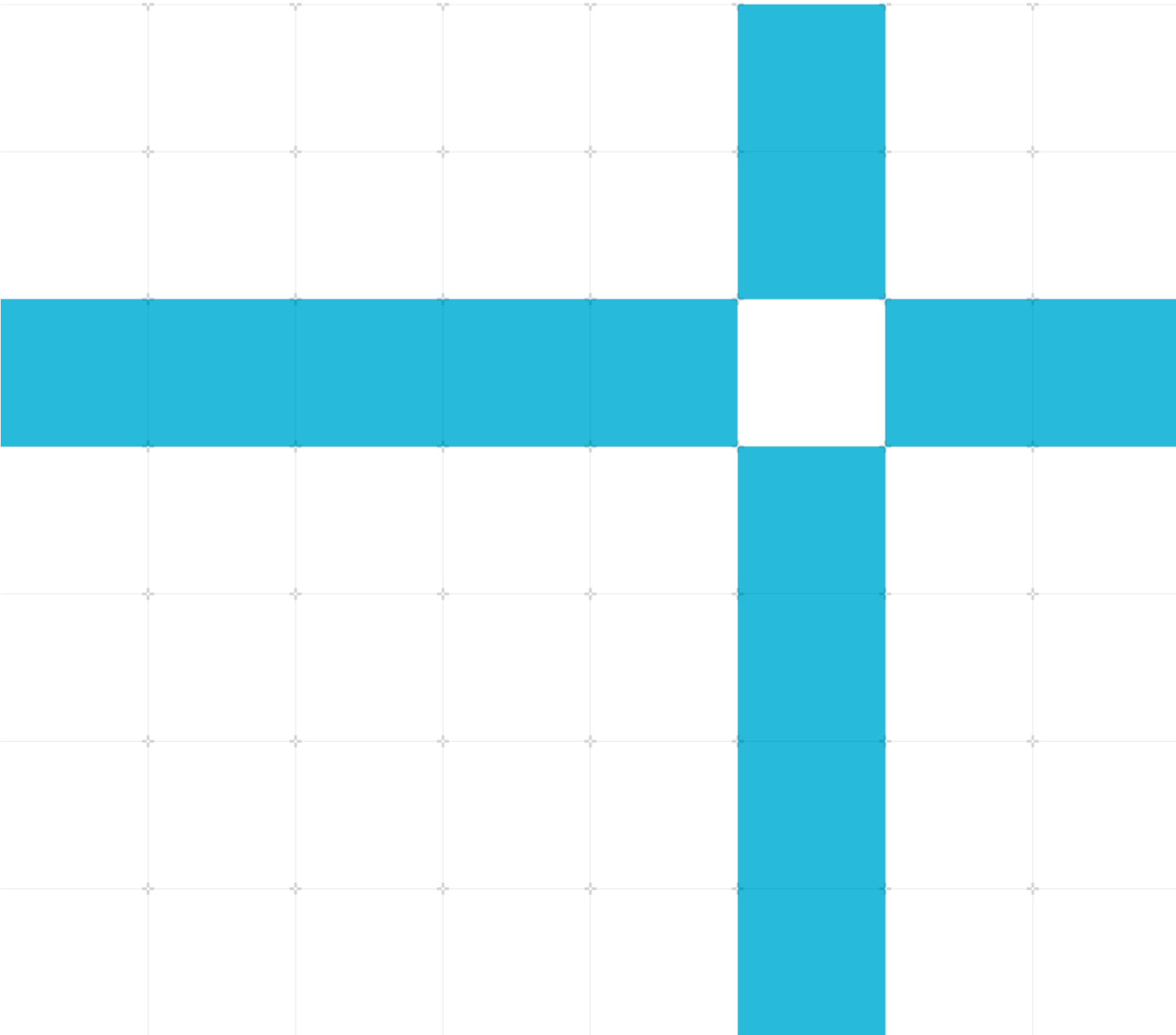
Arm Base Boot Requirements

Revision 2.2

Platform Design Document

Non-Confidential

Issue I
DEN0044_0202_I_en



Arm Base Boot Requirements Platform Design Document

This document (DEN0044_0202_I_en) was issued on June 17, 2025. There might be a later issue at http://developer.arm.com/documentation/DEN0044

The product revision is 2.2.

See also:

- Product and document information
- Useful resources

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on Arm Base Boot Requirements, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey:
https://developer.arm.com/documentation-feedback-survey.

Contents

1. 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 very different. This means that it is impossible to produce a single product that meets the needs of all markets.

The Arm architecture profiles, Application, Real-time, and Microcontroller, provide Arm solutions that align with the functional requirements of different target markets. The differences between products that are targeted at different profiles 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 Trust Zone 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.

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

2. Introduction

This document specifies the base boot requirements for boot and runtime services, based on Arm 64-bit architecture, that system software, for example operating systems (OS) and hypervisors, can rely on.

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 (click the links for more information about relevant documents):

- *Power State Coordination Interface (PSCI)*
- *SMC Calling Convention (SMCCC)*
- *Unified Extensible Firmware Interface Specification (UEFI)*
- *Advanced Configuration and Power Interface Specification (ACPI)*
- *SMBIOS Reference Specification (SMBIOS)*
- *Devicetree Specification (DT) interfaces*

This document specifies requirements that are based on these interfaces. Also, several recipes describe the collections of requirements for the various operating systems and hypervisors.

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

This document is structured as follows:

- Section 3 BBR recipes

  This section provides recipes to accommodate the various operating systems and hypervisors. This includes SBBR, EBBR, and LBBR.

Note SBBR, EBBR and LBBR are the names of the BBR recipes. Section 3 specifies the details of the recipes. The recipes accommodate the various operating systems and hypervisors, regardless of the market segments.

- Section 4 PSCI and SMCCC requirements
- Section 5 Secondary core boot requirements
- Section 6 UEFI requirements
- Section 7 ACPI requirements
- Section 8 SMBIOS requirements

- Section 9 LBBR requirements
- Appendixes

  The Appendixes provide summaries of required and recommended UEFI and ACPI interfaces.

Implementations that are consistent with this document 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.

This document cross-references sections, using the section sign §, in other specifications listed in the Useful resources

Examples:

- ACPI § 5.6.5:
  - Reference to the *Advanced Configuration and Power Interface Specification* section 5.6.5
- UEFI § 6.1:
  - Reference to the *Unified Extensible Firmware Interface Specification* section 6.1

3. BBR recipes

This section provides recipes for various operating systems and hypervisors.

These recipes define the boot and runtime services for a physical system, including services for virtualization.

Note Servers that implement SBBR or LBBR recipes can be compliant with Open Compute Project (OCP) Open System Firmware (OSF) by following the *OCP Open System Firmware Checklist*.

3.1. SBBR recipe

The main goal of this recipe is 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 (BSA) specification*. Sections 4, 5, 6, 7, and 8 specify the requirements that systems using SBBR recipe must meet.

Currently, Windows Server, Windows 11, Windows IoT Enterprise, VMware ESXi, Amazon Linux, and Oracle Linux require SBBR recipe. Other operating systems, for example Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), CentOS, Ubuntu, NetBSD, and FreeBSD can also support SBBR. This list of operating systems and hypervisors might change.

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

3.2. EBBR recipe

Section 4 and in the *Embedded Base Boot Requirements (EBBR) specification* specify the requirements that systems using EBBR recipe must meet.

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 might change.

3.3. 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. Platforms can support LinuxBoot which provides open-source firmware and implements the requirements listed in this section.

Sections 4, 5, 6, 7, and 8 specify the requirements that systems using the LBBR recipe must meet. Section 9 specifies the exceptions.

Various underlying system initialization firmware can implement to LBBR. This can include for example EDK2, coreboot with UEFI shim, or other minimal UEFI firmware.

Note Some hyperscale datacenters use LinuxBoot that is compliant with the *OCP Open System Firmware Checklist*.

3.4. BBR recipes stacks

The following figure shows the SBBR recipe firmware stack.

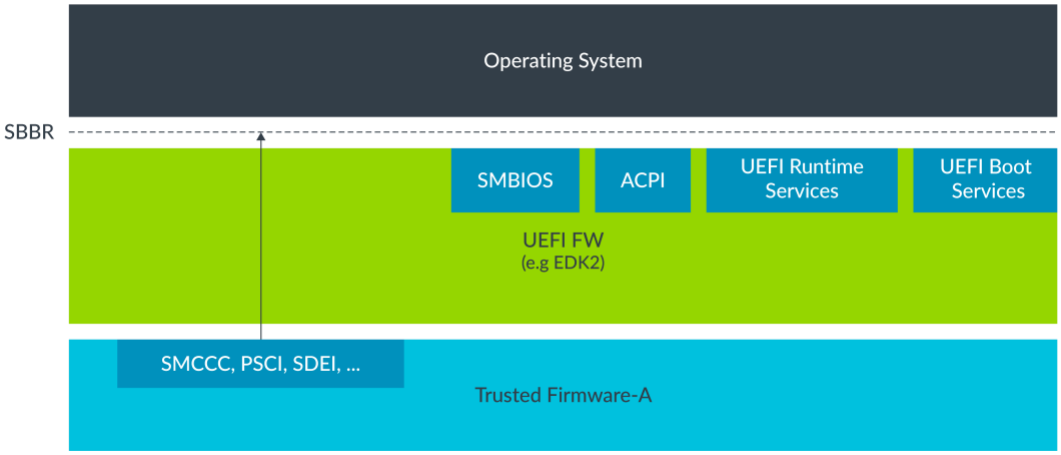


Figure 3-1 SBBR recipe firmware stack example

The following figure shows the EBBR recipe firmware stack.

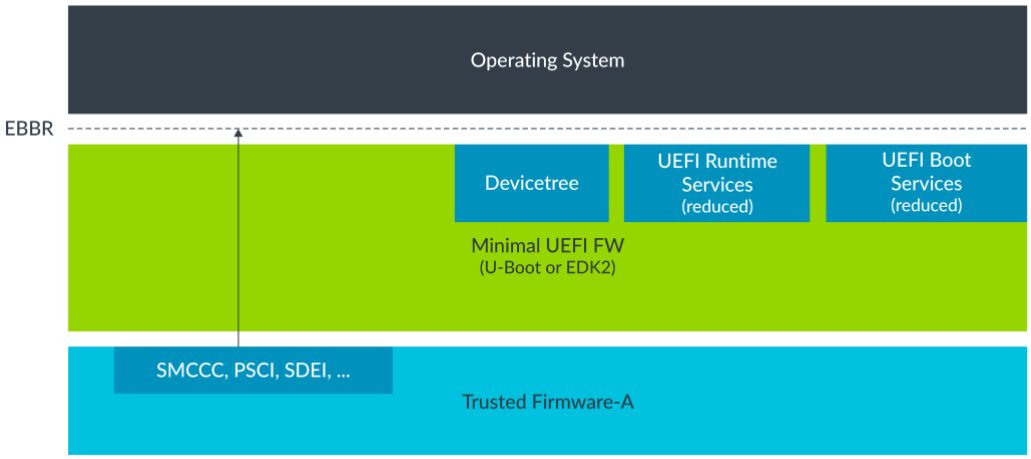


Figure 3-2 EBBR recipe firmware stack example

The following figure shows the LBBR recipe firmware stack examples.

Figure 3-3 LBBR recipe firmware stack examples

4. PSCI and SMCCC requirements

The system must expose *Power State Coordination Interface (PSCI)* and *SMC Calling Convention (SMCCC)* 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 Exception Level 3 (EL3). TF-A implements various Arm interface standards including PSCI and SMCCC.

4.1. SMCCC architecture call requirements

The following table shows the requirements for the SMCCC architecture call.

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 or newer and contain at least one PE that is affected by CVE-2018-3639.
SMCCC_ARCH_WORKAROUND_3	7.7	Recommended for platforms that support SMCCC v1.1 or newer and contain at least one PE that is affected by CVE-2017-5715 or CVE-2022-23960.

Note Only software that is vertically integrated with the SMCCC implementation should call services in the SMCCC vendor-specific EL3 monitor range (FIDs `0x8700_0000—0x8700_FFFF` and `0xC700_0000—0xC700_FFFF`). General purpose EL1 or EL2 software should not call services in this range.

4.2. PSCI call requirements

The system must meet compliance requirements as *Power State Coordination Interface* specification v1.2 § 6.9 defines:

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

Platforms targeted at the PC segment must support the ACPI S4 sleeping state (See Section 7):

- If ACPI S4 sleeping state is supported, the SYSTEM_OFF2 function is recommended to be implemented as specified in PSCI v1.3 Issue F.b or later revisions. The SYSTEM_OFF2 function is a dedicated PSCI command to hibernate the system.

4.3. Other SMC call requirements

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

4.3.1. True random number generator firmware interface

A system can implement an entropy source in secure world privileged software. If the system exposes the entropy source to the normal world, for example, operating systems, via an SMCCC interface, it must use the Arm True Random Number Generator (TRNG) firmware interface as defined in the *Arm True Random Number Generator Firmware Interface* specification.

4.3.2. Platform firmware architectural feature support

Platform firmware must handle any traps associated to the following list of features, if that feature is present in the system:

- FEAT_FGT
- FEAT_FGT2
- FEAT_PAuth

Platform firmware must deploy the Undef Injection procedure when handling unknown traps.

Undef Injection: the trap handler sets the system registers of the trap originating EL such that, on ERET to that EL, an exception with EC=0 (Unknown reason) is raised.

5. Secondary core boot requirements

Platforms providing EL3 must implement the power state coordination interface:

Power State Coordination Interface (PSCI) v1.2 Published July 2022

This interface is 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 specify low power state entry methods and low power state residency and usage statistics. See *Arm Functional Fixed Hardware* specification for more details.

Note Platforms targeted at the PC segment are recommended to implement both the platform-coordinated mode and the OS-initiated mode of power state coordination. Both of these modes are defined in PSCI, Some Operating Systems only support the OS-initiated mode.

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 must 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.

6. UEFI requirements

6.1. UEFI version

This document references the following specification and versions:

Unified Extensible Firmware Interface Specification (UEFI) v2.11 Published December 2024, includes the AArch64 bindings

6.2. UEFI compliance

Any UEFI-compliant system must follow the requirements that section 2.6 of the UEFI specification describes. 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 must be implemented. The network boot technology protocols include PXE, HTTP(s), and/or iSCSI.

6.3. UEFI system environment and configuration

6.3.1. 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.

6.3.1.1. UEFI boot at EL2

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

6.3.1.2. UEFI boot at EL1

A guest operating system environment can boot UEFI at EL1, 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.

6.3.2. System volume format

The system firmware must support the disk partitioning schemes that the UEFI specification requires. See UEFI § 2.6.2 and UEFI § 13.3.1.

6.3.3. UEFI image format

UEFI enables the extension of platform firmware, by loading UEFI driver and UEFI application images. See UEFI § 2.1.

6.3.3.1. UEFI drivers

A device can provide a container for one or more UEFI drivers that are used for the device initialization. See UEFI § 2.1.4. 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.

6.3.3.2. UEFI applications

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

6.3.3.3. PE/COFF image

The SectionAlignment and FileAlignment fields in Microsoft PE Format must contain the value of at least 4KiB. Higher values are possible. For example, UEFI specification requires `DXE_RUNTIME_DRIVER` modules to have the 64KiB granular memory type.

Modules that 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 possible if the section alignment and file alignment are equal.

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

6.3.4. GOP protocol

For systems with graphics video hardware, `EFI_GRAPHICS_OUTPUT_PROTOCOL` 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.

6.3.5. Address translation support

If a platform includes PCI bus support, then the `EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL` and the `EFI_PCI_IO_PROTOCOL` 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.

6.4. UEFI boot services

6.4.1. 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 that the UEFI memory map describes must be identity-mapped. This means that virtual addresses must have equal physical addresses.

The default RAM allocated attribute must be `EFI_MEMORY_WB`.

6.4.2. UEFI-loaded images

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

6.4.3. 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 6.6 or later with a Hardware-Reduced ACPI model.
    See section 7.
- `SMBIOS3_TABLE_GUID`
  - This table defines the 64-bit entry point for SMBIOS table.
  - The SMBIOS tables must conform to version 3.8.0 or later of the *SMBIOS Reference Specification*. See section 8.

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.
  - 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 in this optional devicetree:
    - /chosen/linux,initrd-start
    - /chosen/linux,initrd-end

6.5. 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.

6.5.1. 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.

6.5.2. 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 enable 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 64KiB page.

6.5.3. Real-time clock

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

- GetTime()
- SetTime()

SetTime() can return an error on systems where the real-time clock cannot be set by this call.

6.5.4. 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 call into PSCI. The following table maps the UEFI and PSCI reset calls:

EFI ResetSystem() ResetType	PSCI Reset call
EfiResetShutdown	SYSTEM_OFF
EfiResetCold	SYSTEM_RESET
EfiResetWarm	SYSTEM_RESET2, with reset_type = 0x0 (SYSTEM_WARM_RESET) If SYSTEM_RESET2 is not implemented, SYSTEM_RESET is used.
EfiResetPlatformSpecific, with a platform-specific ResetData GUID.	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 might call either interface to reset the system. Calling the UpdateCapsule() service requires the use of the UEFI runtime service for reset.

6.5.5. 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.

6.6. Firmware update

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

`EFI_UPDATE_CAPSULE` is not required if an out-of-band facility such as an independent Baseboard Management Controller (BMC) performs the firmware update.

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

The `EFI_UPDATE_CAPSULE` implementation is 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 UEFI 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 the *Arm Architecture Reference Manual for A-profile architecture* (Arm ARM) v8.2 or Cache Clean to Point of Deep Persistence introduced in Arm ARM v8.5. This is required only when UpdateCapsule() is called after ExitBootServices(). This requirement is clarified in UEFI § 8.5.3.

7. ACPI requirements

7.1. ACPI version

This document references the following specification and versions:

Advanced Configuration and Power Interface Specification (ACPI) 6.6 Published May 2025

ACPI describes the hardware resources that are installed, and 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.

7.2. ACPI provided data structures

All platforms that comply with this specification must:

- Conform to the ACPI specification, version 6.6 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).

7.3. 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 6.6 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.

7.3.1. Mandatory ACPI tables

The following tables are mandatory for all compliant systems.

7.3.1.1. RSDP

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

7.3.1.2. 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 uses.

7.3.1.3. 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 in ACPI § 5.2.9.1 is selected.
  - The ARM_BOOT_ARCH flags describe the presence of PSCI. See ACPI § 5.2.9.4.
  - The SLEEP_CONTROL_REG and SLEEP_STATUS_REG must evaluate to zero.

7.3.1.4. 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.

7.3.1.5. 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 increases in hardware thread, then cluster, and then socket ordering:
  - 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

7.3.1.6. GTDT

- Generic Timer Descriptor Table (GTDT), ACPI § 5.2.25.
  - This table describes the Arm Generic Timer block and the Arm Generic Watchdog.

7.3.1.7. DBG2

- Debug Port Table 2 (DBG2). See **http://uefi.org/acpi**
  - This table provides a standard debug port.
  - This table is required to be published only when serial debug is present and enabled on the system.

- 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 7.3.1.8).
- 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. If only one UART is available in the system, it must be configured either as console or debug UART, and not combined.

7.3.1.8. SPCR

- Serial Port Console Redirection (SPCR). See http://uefi.org/acpi
  - This table is required to be published only when serial console redirection is present and enabled on the system.
  - This table provides the essential configuration information that is needed for headless operations, like a serial console.
  - This table defines a serial port type, location, and interrupts.
  - This table can 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.
- Revision 4 of the SPCR table is recommended to be implemented. Revision 4 adds the following:
  - Support for describing the UART clock frequency and precise baud rate, allowing for configuring non-traditional baud rates used with high speed UARTs.
  - Support for a NamespaceString[] to help correlate the console device described in the SPCR with the DSDT entry.
- The SPCR must have correct ACPI GSIV interrupt routing information for the UART device.
- The UART console device is recommended to be configured with a default baud rate of 115200.
- The SPCR console device must be in the ACPI namespace, as described in *ACPI for Arm Components specification* § 2.3 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)

7.3.1.9. MCFG

- PCI Memory-mapped Configuration Space (MCFG). *PCI Firmware Specification* (PCI FW) version 3.3 § 4.1.2
  - This table described the PCIe ECAM base address.
  - This table is required if PCIe is supported.
  - Multiple PCIe ECAM regions per PCI Segment are allowed. See *PCI Firmware Specification* for more details.

7.3.1.10. PPTT

- Processor Properties Topology Table (PPTT), ACPI § 5.2.31.
  - This table describes the topological structure of processors that are controlled by the OSPM and their shared resources.

7.3.2. Recommended ACPI tables

Appendix E listed the recommended ACPI tables.

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.

7.3.3. Optional ACPI tables

All other tables in the *Advanced Configuration and Power Interface Specification* can be used as needed for AArch64 platforms, but only if they comply with syntax and semantics of the specification.

7.4. ACPI definition blocks

Within the DSDT or SSDT tables that describe the platform, ACPI definition blocks describes devices. 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. Processors is an example of such devices. However, PCI devices are enumerated by a defined protocol.

7.5. 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. ACPI specification predefines standard objects and methods. Platform designers can define vendor-specific objects and methods as needed.

All objects and methods must conform to the definitions in *Advanced Configuration and Power Interface Specification* version 6.6 or later. Legacy definitions are not supported.

Appendix F lists the recommended ACPI methods. 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.

7.5.1. 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

7.5.2. 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 6.6 or later of the *Advanced Configuration and Power Interface Specification*:

- _ADR: Address on the parent bus of the device. A device in the ACPI name space must have either this object or the _HID object. This object is essential for PCI(e) devices in the ACPI name space, for example.
  - See ACPI § 6.1.1
- _CCA: Cache Coherency Attribute. This object provides information about whether a device must 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 § 6.2.18
- _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. A device in the ACPI name space must have either this object or the _ADR object.
  - 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.

7.5.3. 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.

7.5.4. 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 the interrupts that are 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.

7.5.5. Address translation support

7.5.5.1. Legacy I/O Space

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.

7.5.5.2. PCIe 32-bit non-prefetchable to 64-bit MMIO space mapping

If the platform supports remapping of 32-bit non-prefetchable MMIO space to high memory (> 4GiB) space, then the translation offset for the remapping must be in the _TRA subfield of the ACPI _CRS object that describes this MMIO space. The device object in ACPI namespace that describes the PCIe host bridge must contain this _CRS.

Because non-ACPI aware software cannot access the ACPI _TRA field in the _CRS, it requires a protocol stack implementation to handle the address translation for a remapped address. For example, UEFI applications and drivers must use the protocol stack, `EFI_PCI_IO_PROTOCOL`, GetBarAttributes to get the host side address for any direct BAR accesses instead of reading the PCI configuration space. Any Graphics Output Protocols used must have the host address in the FrameBufferBase address. It is recommended that only accesses that require translation use the protocol stack to reduce firmware and OS access differences.


Note Non-UEFI software might require an implementation defined method to access remapped addresses.


Warning Some drivers and devices like graphics cards might rely on 32-bit MMIO and might not comprehend remapped memory. The requirements above might not be easily enforced on some systems. 32-bit MMIO might need to be reserved for specific devices in some use cases.

7.5.6. Describing Arm components in ACPI

ACPI for Arm Components specification defines special ACPI properties and IDs for certain components that Arm implements or licenses. If a platform includes any of the Arm components that are listed in the specification, it must follow the guidance of that specification for implementing the ACPI objects and methods for such components.

7.5.7. Power management

7.5.7.1. Hibernate

Platforms targeted at the PC segment must support the ACPI S4 sleeping state.

If ACPI S4 sleeping state is supported, the _S4 object must be presented, and it must evaluate to 0.

7.5.7.2. Power source

Battery-powered devices targeted at the PC segment must implement one ACPI Power Source device to describe the system's power sources.

If there are multiple physical power supply sources in a battery-powered device targeted at the PC segment, then only a single ACPI Power Source device must be used to represent all the physical power supply sources in the system.

ACPI Power Source device, if present, must implement the objects listed in Appendix F, under "Power Source".

7.5.7.3. Battery

Battery-powered devices targeted at the PC segment must implement at least one ACPI Control Method Battery device.

ACPI Control Method Battery, if present, must respond with notification codes `0x80` and `0x81` for all appropriate events, and must implement the objects listed in Appendix F, under "Battery".

7.6. Hardware requirements imposed on the platform by ACPI

The term HW-Reduced does not imply anything about functionality. HW-Reduced means that the hardware specification is not implemented. See Chapter 4 of the *Advanced Configuration and Power Interface 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-signaled events or GPIO-signaled events must be used to generate interrupts that are functionally equivalent to General Purpose Events (GPEs). See ACPI § 5.6.4, §5.6.5, and §5.6.9.

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. See ACPI § 18.3.2.7.2

- *Software Delegated Exception Interface* (SDEI) 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 the power button.

7.6.1. Processor performance control

If the platform supports OSPM-directed processor performance control, using Collaborative Processor Performance Control (CPPC) to expose this feature is mandatory.

CPPC version 3 or higher must be implemented, as indicated by the Revision field of the _CPC (Continuous Performance Control) object.

See Bit[14] of Table 6-13 Platform-Wide _OSC Capabilities, ACPI § 6.2.12.2.


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 Functional Fixed Hardware* (FFH) specification for more details.

The Energy Performance Preference Register, if implemented, must be at least 8-bits wide.

The nominal frequency supported by the processor must be indicated, and the corresponding fields of the _CPC object must not evaluate to 0.

It is recommended that any of the following registers, if implemented, exist in the System Memory address space for market segments where it is beneficial for OSPM to access these registers with low latency:

- Desired Performance Register
- Maximum Performance Register
- Minimum Performance Register
- Energy Performance Preference Register

7.6.2. Time and alarm device

If the ACPI time and alarm device is implemented (see ACPI § 9.17), it must operate on the same real-time clock that is exposed by the UEFI runtime services.

8. SMBIOS requirements

The *SMBIOS Reference Specification* is published by the DMTF, SMBIOS is an important firmware component for computer systems. It 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.

8.1. SMBIOS version

This document references the following specification and versions:

SMBIOS Reference Specification v3.8.0 Published August 2024.

Legacy SMBIOS tables and formats are not supported.

8.1.1. 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.

8.2. 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, see 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 more guidance and requirements on SMBIOS tables reporting for telemetry and servicing, see *Firmware Windows Engineering Guide*.

8.2.1. Type00: BIOS Information

The type is required with the following fields mandatory.

- 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

8.2.2. Type01: System Information

The type is required with the following fields mandatory:

- 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.

8.2.3. Type02: Baseboard or Module Information

This type is recommended:

- Manufacturer
- Product
- Version

- Serial Number
- Asset Tag
- Location in Chassis
- Board Type

8.2.4. Type03: System Enclosure or Chassis

The type is required with the following fields mandatory:

- 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

8.2.5. Type04: Processor Information

The type is required with the following fields mandatory:

- 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.
  - For systems that support SMBIOS 3.8.0 or newer, and support SMCCC_ARCH_SOC_ID call with parameter SoC_ID_type == 2, this field must implement the SoC name string, as specified by SMBIOS specification.

- External Clock
- Max Speed
- Current Speed
- Status
- Processor Upgrade
- Serial Number
- Asset Tag
- Part Number
- Core Count
- Core Enabled
- Thread Count
- Processor Characteristics
- Processor Family 2
- Core Count 2
- Core Enabled 2
- Thread Count 2
- Thread Enabled
- Socket Type

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. The chip package implements a chip-to-chip extension of cache coherency, and typically participates within the same Inner Shareable domain. See Inner Shareable domain as defined in *Arm Architecture Reference Manual for A-profile architecture*.

8.2.6. Type07: Cache Information

The type is required with the following fields mandatory:

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

8.2.7. Type08: Port Connector Information

This type is recommended for platforms with physical ports:

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

8.2.8. Type09: System Slots

This type is required with the following fields mandatory 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.

8.2.9. Type11: OEM Strings

This type is recommended:

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

8.2.10. Type13: BIOS Language Information.

The type is recommended:

- Installable Languages

- Flags
- Current Language

8.2.11. Type14: Group Associations

The type is 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).

8.2.12. Type16: Physical Memory Array

The type is required with the following fields mandatory:

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

8.2.13. Type17: Memory Device

The type is required with the following fields mandatory:

- 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

8.2.14. Type19: Memory Array Mapped Address

The type is required with the following fields mandatory:

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

8.2.15. Type32: System Boot Information

The type is required with the following fields mandatory.

- Boot Status

8.2.16. Type38: IPMI Device Information

The type is required with the following fields mandatory for platforms with IPMI v1.0 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 *Intelligent Platform Management Interface* (IPMI) v1.5 and v2.0.

8.2.17. Type39: System Power Supplies

The type is 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.

8.2.18. Type41: Onboard Devices Extended Information

The type is recommended:

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

8.2.19. Type42: Redfish Host Interface

The type is required with the following fields mandatory for platforms supporting *Redfish Host Interface Specification*:

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

8.2.20. Type43: TPM Device

The type is required with the following fields mandatory for platforms with a TPM:

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

8.2.21. Type44: Processor Additional Information

The type is recommended:

- Processor-Specific Block
  - Processor-Specific Data

Processor-Specific Block is defined with the following *Processor-Specific Data subtypes:*

- 0 - AArch64 Architecture data
- 1 – Vendor-specific data

Subtype *0 - AArch64 Architecture data* is recommended to be implemented.

Systems that provide vendor-specific additional processor data (using subtype *1 – Vendor-specific data)* are recommended to implement separate instances of SMBIOS Type 44 for sub-types 0 and 1.

8.2.22. Type45: Firmware Inventory Information

The type is recommended:

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

This SMBIOS type describes firmware components in the system, such as TF-A, SCP, and NIC firmware. 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
  - Can 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
  - Can 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
  - Can be an entry per PCIe option device (such as a NIC) that presents a firmware version via UEFI Firmware Management Protocol (FMP).

8.2.23. Type 46: String Property

The type is 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.

9. LBBR requirements

9.1. LBBR requirements

The system firmware for LBBR recipe must meet the requirements specified in this section. With this approach, LinuxBoot kernel exposes the required interfaces, UEFI Runtime, ACPI, and SMBIOS, to the final runtime Linux kernel. LinuxBoot kernel uses `kexec` to load the final Linux kernel. 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.

9.1.1. ACPI

The system firmware must meet all the requirements in section 7, ACPI, with no exceptions.

9.1.2. SMBIOS

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

9.1.3. UEFI

The system firmware must meet all the requirements in section 6, UEFI, with exceptions, as the following table shows:

BBR §	LBBR requirements exception	Notes
6.1 UEFI version	No	All requirements apply.
6.2 UEFI compliance	Yes	LBBR defines several exceptions for the requirements in Section 2.6 of the UEFI specification and Appendix A, Appendix B, and Appendix C of this document. For details of the exceptions, refer to 9.1.3.1.
6.3.1 AArch64 Exception Levels	No	All requirements apply.
6.3.2 System volume format	Yes	Not applicable for LBBR.
6.3.3.1 UEFI image format – UEFI drivers	Yes	Not applicable for LBBR.
6.3.3.2 UEFI image format – UEFI applications	Yes	Not applicable to LBBR.
6.3.3.3 UEFI image format – PE/COFF image	Yes	Requirements apply only to the LinuxBoot kernel image, using EFI Stub.

BBR §	LBBR requirements exception	Notes
6.3.4 GOP Protocol	Yes	These requirements are optional for LBBR compliant systems that wish to pass the UEFI video frame buffer to the final operating system.
6.3.5 Address translation support	Yes	Not applicable for LBBR.
6.4.1 UEFI boot services - memory Map	No	All requirements apply.
6.4.2 UEFI Loaded Images	Yes	Requirements apply only to the LinuxBoot kernel image, using EFI Stub.
6.4.3 Configuration tables	No	All requirements apply.
6.5.1 Runtime Exception Level	No	All requirements apply.
6.5.2 Runtime memory map	No	All requirements apply.
6.5.3 Real-time clock	Yes	All UEFI time runtime services are optional for LBBR.
6.5.4 UEFI reset and shutdown	Yes	All UEFI reset and shutdown services are optional for LBBR. If implemented, the requirements of mapping the UEFI runtime services to PSCI apply.
6.5.5 Set variable	Yes	All UEFI variables runtime services are optional for LBBR.
6.6 Firmware Update	Yes	In-band firmware update using UEFI capsule runtime services are optional for LBBR
Appendix A Required UEFI Boot Services	Yes	For details of UEFI Boot Services requirements and exceptions, see 9.1.3.2.
Appendix B Required UEFI Runtime Services	Yes	For details of UEFI runtime services requirements and exceptions, see 9.1.3.3.
Appendix C Required UEFI Protocols	Yes	For details of UEFI Protocols requirements and exceptions, see 9.1.3.4.
Appendix D Optional UEFI Protocols	Yes	None of the UEFI Protocols outlined as optional are applicable to LBBR.

9.1.3.1. Deviation from UEFI Specification requirements

LBBR compliant systems must conform to a subset of the UEFI Specification as this section describes. 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 requirements exception	Notes
2.6.1 - Required Elements	Yes	`EFI_SYSTEM_TABLE`, `EFI_BOOT_SERVICES`, and `EFI_RUNTIME_SERVICES` are required for LBBR systems. All UEFI System Table function pointers, including the UEFI Runtime Services, must be available and cannot be NULL pointers. Some UEFI runtime services can return `EFI_UNSUPPORTED`, as detailed in the requirements in section 9.1.3.2 and 9.1.3.3. 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. All other elements are either not required or are not applicable to LBBR.
2.6.2 - Platform Specific Elements	Yes	None of the platform specific UEFI elements, including all UEFI protocols, are required or applicable to LBBR.
3.3 - Globally Defined Variables	Yes	None of the Globally Defined UEFI Variables are required for LBBR systems. **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 might require platform specific methods to provide the equivalent functionality.

9.1.3.2. UEFI boot services requirements

UEFI boot services exist before the call to ExitBootServices() and provide a limited set of functions to be used in a preboot environment, and by OS loaders. A limited subset of functions contained in `EFI_BOOT_SERVICES` are required to boot. The system must provide a system memory map, which must include all appropriate devices and memories for booting and system configuration. The system must provide the facilitation for `EFI_RNG_PROTOCOL` to enable a hardware source of entropy for Kernel Space Layout Randomization (KASLR).

Note All functions defined as boot services must exist. Methods for unsupported or unimplemented behavior must return an appropriate error code.

EFI_BOOT_SERVICES function	LBBR requirements exception	Notes
AllocatePages	No	Memory Services, allocates pages of a particular type.

EFI_BOOT_SERVICES function	LBBR requirements exception	Notes
FreePages	No	Memory Services, frees allocated pages.
GetMemoryMap	No	Memory Services, returns the current boot services memory map and memory map key.
AllocatePool	No	Memory Services, allocates a pool of a particular type.
FreePool	No	Memory Services, frees allocated pool.
HandleProtocol	No	Protocol Handler Services, queries a handle to determine if it supports a specified protocol.
LocateDevicePath	Yes	Protocol Handler Services, locates all devices on a device path that support a specified protocol and returns the handle to the device that is closest to the path. Conditionally required if `EFI_LOAD_FILE2_PROTOCOL` is implemented.
InstallConfigurationTable	No	Protocol Handler Services, adds, updates, or removes a configuration table from the EFI System Table.
ExitBootServices	No	Image Service, terminates boot services. After a successful call to ExitBootServices() all boot services are not callable anymore.
LocateProtocol	No	Library Service, finds the first handle in the handle database the supports the requested protocol. Used by Linux to get protocol interfaces, including `EFI_RNG_PROTOCOL` which is needed for KASLR.
InstallMultipleProtocolInterfaces	Yes	Library Services, installs one or more protocol interfaces onto a handle. Conditionally required if `EFI_LOAD_FILE2_PROTOCOL` is implemented.
UninstallMultipleProtocolInterfaces	Yes	Library Services, uninstalls one or more protocol interfaces onto a handle. Conditionally required if `EFI_LOAD_FILE2_PROTOCOL` is implemented.
CopyMem	No	Miscellaneous Service, copies the contents of one buffer to another buffer.
SetMem	No	Miscellaneous Service, fills a buffer with a specified value.
GetNextMonotonicCount	Yes	Implementation not required, but if not implemented, the function must return `EFI_DEVICE_ERROR` as `EFI_UNSUPPORTED` is not an allowed status code.
LoadImage	Yes	Loads an EFI image into memory. Conditionally required if `EFI_LOAD_FILE2_PROTOCOL` is implemented.
StartImage	Yes	Loads an EFI image into memory. Conditionally required if `EFI_LOAD_FILE2_PROTOCOL` is implemented.

EFI_BOOT_SERVICES function	LBBR requirements exception	Notes
All other Boot Services	Yes	Other Boot services are optional, but their functions must return an appropriate error code if not implemented or supported.

9.1.3.3. UEFI runtime services requirements

UEFI runtime services exist after the call to ExitBootServices() and 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, `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. Also, some of these functions might not be available after ExitBootServices() is called. In that case, the firmware provides the `EFI_RT_PROPERTIES_TABLE` to show which functions are available during runtime services. Functions that are not available during runtime services return `EFI_UNSUPPORTED`.

The table below shows 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

9.1.3.4. UEFI Protocols requirements

The table below shows which UEFI protocols are required to be implemented.

UEFI Protocol	LBBR requirements exception	Notes
EFI_RNG_PROTOCOL	No	EFI_RNG_PROTOCOL is required to enable a hardware source of entropy for Kernel Space Layout Randomization (KASLR)
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL	Yes	Implementation of the OutputString is recommended for early boot console printing, which is useful for various debugging purposes. All other parameters are optional but return EFI_UNSUPPORTED or EFI_DEVICE_ERROR as defined in UEFI § 12.4.
EFI_LOAD_FILE2	Yes	This Protocol is recommended because it is used by the Linux zboot on Arm to boot a compressed kernel image.
All other protocols	Yes	All other Protocols are 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

Service	UEFI §
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 gets files from arbitrary devices that are primarily boot options. This includes, for example, booting from network devices, The Load File 2 protocol gets files from arbitrary devices that are not boot options. The Media I/O protocols must 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

Operating systems use the EFI_RNG_PROTOCOL 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, this protocol implementation might be backed by the *Arm True Random Number Generator Firmware Interface*.

Appendix D. Optional and conditionally required UEFI protocols

This section lists optional UEFI protocols. These might 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	28.1.3
EFI_TCP4_SERVICE_BINDING_PROTOCOL	28.1.1
EFI_IP4_SERVICE_BINDING_PROTOCOL	28.3.1
EFI_IP4_CONFIG2_PROTOCOL	28.4.1
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	29.3.3
EFI_DHCP6_SERVICE_BINDING_PROTOCOL	29.3.1
EFI_TCP6_PROTOCOL	28.2.3
EFI_TCP6_SERVICE_BINDING_PROTOCOL	28.2.1
EFI_IP6_SERVICE_BINDING_PROTOCOL	28.5.1
EFI_IP6_CONFIG_PROTOCOL	28.6
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	29.7.7
EFI_REST_EX_SERVICE_BINDING_PROTOCOL	29.7.8
EFI_REDFISH_DISCOVER_PROTOCOL	31.1.4
EFI_REST_JSON_STRUCTURE_PROTOCOL	29.7.17.2

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	29.6.10
EFI_HTTP_BOOT_CALLBACK_PROTOCOL	24.7.11
EFI_DNS4_SERVICE_BINDING_PROTOCOL	29.4.1

Service	UEFI §
EFI_DNS4_PROTOCOL	29.4.2
EFI_DNS6_SERVICE_BINDING_PROTOCOL	29.5.1
EFI_DNS6_PROTOCOL	29.5.3
EFI_TLS_SERVICE_BINDING_PROTOCOL	28.9.1
EFI_TLS_PROTOCOL	28.9.2
EFI_TLS_CONFIGURATION_PROTOCOL	28.9.7

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) Specification* 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 § 9.11.3), the System Firmware enumeration (CXL § 9.11.5), and the CXL device discovery flow (CXL § 9.11.6)
- 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 § 9.18)

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 can 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 DEN0049

If an I/O subsystem supports I/O coherency, configure 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, configure 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 *Arm Base System Architecture* and "CCA: Cache Coherent Attribute" in *IO Remapping Table* specification.

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 DEN0054
AEST	Arm Error Source Table	https://uefi.org/acpi DEN0085
RAS2	ACPI RAS2 Feature Table	5.2.21

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	5.2.29

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

PDTT describes one or more PCC subspace identifiers that can trigger and notify the platform specific debug facilities to capture non-architectural system state. This is a standard mechanism for the OSPM to notify the platform of a fatal crash, for example kernel panic or bug check.

ACPI signature	Full name	ACPI §
PDTT	Platform Debug Trigger Table	5.2.30

NVDIMM Firmware Interface

NFIT describes NVDIMM if NVDIMM is supported.

ACPI signature	Full name	ACPI §
NFIT	NVDIMM Firmware Interface Table	5.2.26

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	5.2.23

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 table is required to support CXL host bridges in the operating system. The operating system uses 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 follow the guidelines outlined in the *ACPI for Arm MPAM* Specification.

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 DEN0065

iSCSI

Support for iSCSI is only required on machines that lack persistent storage, like an HDD. This configuration is 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 DEN0117

Diagnostic Dump Device

The AGDI Table in the *ACPI for Arm Components* specification describes a generic diagnostic dump device that can 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 DEN0093

Boot Performance

The FPDT Table is a top-level ACPI table pointing to sub-tables that hold firmware performance (timestamp) figures.

ACPI signature	Full name	ACPI §
FPDT	Firmware Performance Data Table	5.2.24

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. If the platform supports OSPM-directed processor performance control, the use of CPPC is required.

Method	Full name	ACPI §
_CPC	Continuous Performance Control (replaces _PCT and _PSS)	8.4.6.1
_PSS	Performance Supported States (Superseded by _CPC)	8.4.5.2

CPU and system idle control

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

Method	Full name	ACPI §
_LPI	Low Power Idle States	8.4.3

NUMA

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

Method	Full name	ACPI §
_PXM	Proximity	6.2.15
_SLI	System Locality Information	6.2.16
_HMA	Heterogeneous Memory Attributes	6.2.19

IPMI (*Intelligent Platform Management Interface*)

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

Method	Full name	§
_IFT	IPMIv2: the IPMI Interface Type, if IPMI has been implemented	ACPI § 5.5.2.4.5, IPMI spec
_SRV	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 (Device Specific Data) Implementation Guide.	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.	6.1.7
_STR	Device description in a single language. Superseded by _MLS.	6.1.10


Note

_MLS is preferred over _STR.

CXL

The following methods are required to support CXL host bridges in the operating system. The operating system might 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 § 9.18.2

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. See *ACPI for CoreSight* specification.	https://uefi.org/acpi DEN0067

Time and Alarm

Method	Full name	ACPI §
_GCP	Get the capabilities of the time and alarm device	9.17.2
_GRT	Get the Real time	9.17.3
_SRT	Set the Real time	9.17.4
_GWS	Get Wake status	9.17.5
_CWS	Clear Wake Status	9.17.6
_STP	Sets expired timer wake policy for the specified timer.	9.17.7
_STV	Sets the value in the specified timer.	9.17.8
_TIP	Returns the current expired timer policy setting of the specified timer	9.17.9
_TIV	Returns the remaining time of the specified timer	9.17.10

SDEI

Devices that use SDEI-based event signaling must use the _DSM method recommended in Appendix E of *Software Delegated Exception Interface* specification, to describe the event number being used for the signaling.

Method	Full name	§
_DSM	SDEI _DSM	SDEI § E DEN0054

PCI and PCIe

Method	Full name	PCI FW §
_CBA	Memory-mapped Configuration Base Address	4.1.3
_DSM	_DSM definitions for PCI	4.6
_OSC	PCI Express capabilities	4.5

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 *PCI Firmware Specification*.
- If the firmware reserves memory regions using Reserved Memory Range nodes in the *IO Remapping Table*, 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.

Device Power Management

The following objects are recommended to be implemented for all devices and power resources which need to be managed by the operating system during device D-state or ACPI system sleep state transition. The implementation can choose to provide either all or a subset of these objects according to system topology and supported device and system power states.

Method	Full name	ACPI §
_PSx	Power State Dx: Control method used to transition the device into Dx power state.	7.3
_SxD	Sx Device State: The shallowest D-state supported by the device in the Sx state.	7.3
_PRx	Power Resources for Dx: List of power resources upon which this device is dependent when it is in the Dx power state.	7.3
_ON	Power resource control method to put the power resource into the ON state.	7.2.3
_OFF	Power resource control method to put the power resource into the OFF state.	7.2.2
_STA	Power Resource Status: Returns the current ON or OFF status for the power resource	7.2.4
_SxW	Sx Device Wake State: Deepest D-state supported by this device in the Sx system sleeping state that can wake the system. Applicable for wake capable devices only.	7.3
_PRW	Power Resources for Wake: List of power resources upon which the device depends for wake. Applicable for wake capable devices only.	7.3.13
_DSW	Device Sleep Wake: Control method used to enable or disable the device's ability to wake a sleeping system. Applicable for wake capable devices only.	7.3.1

Power Source

The following methods must be implemented if the ACPI Power Source device is present.

Method	Full Name	ACPI §
_PSR	Power Source: Control method used to determine if the power source is offline or online.	10.3.1

Battery

The following methods must be implemented if the ACPI Control Method Battery is present.

Method	Full name	ACPI §
_BIX	Battery Information Extended: Returns extended static information about the battery.	10.2.2.3
_BST	Battery Status: Returns the current battery status.	10.2.2.11

Method	Full name	ACPI §
_BTP	Battery Trip Point: Sets the Battery Trip point which generates an event to issue notification (0x80) when battery capacity reaches the specified point.	10.2.2.14
_STA	Device Status: Indicates battery presence, even in removable or docked batteries.	6.3.7

Appendix G. CXL requirements

G.1. CXL host bridge

For each Compute Express Link (CXL) host bridge in the system, the firmware must produce a CXL Host Bridge Structure (CHBS) 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 in the *Compute Express Link (CXL) Specification*.

G.1.1. ACPI device object for CXL host bridge

Each CXL host bridge in the system must be in ACPI namespace as a device object with the CXL host bridge specific HID value of "ACPI0016". See CXL Host Bridge Structure (CHBS), CXL §9.18.1.2. The device object must minimally include a unique _UID, apart from the HID. If the host bridge is also in the CEDT using a dedicated CHBS structure, the _UID value of the device object must match the UID field of the CHBS structure.

G.2. CXL root device

A CXL root device is the origin of a hierarchy of CXL Host-managed Device Memory (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.
- Enable interleaving across CXL host bridges
- Specify 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

The firmware must create a single instance of the ACPI CXL root device object in ACPI namespace to advertise the properties of the CXL root device to the OS. The CXL root device is a single device recognized by its HID value of "ACPI0017". The CXL root device is described in detail in the CXL specification.

The firmware must include the following methods as children of the CXL 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, it must present the CDAT table for NUMA properties of the HDM. Also, if presented to the OS, the UEFI memory map, and the HMAT, SRAT and CEDT tables must use the information obtained from the CDAT tables.

If CXL-attached HDM is present in a system and its CDAT DSEMTS structure indicates that the HDM is normal memory, the HMAT table is recommended to describe the NUMA properties of the HDM.

Note Normal memory can be either EfiConventionalMemory or EfiConventionalMemory with the EFI_MEMORY_SP attribute set.

The HMAT table is recommended to be alongside with an SRAT table to describe the HDM. The SRAT table is a prerequisite for the HMAT table.

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 and NUMA properties of the path between the PEs and the CXL gateway/bridge associated with each CXL host bridge. The Generic Port entry is recommended to show 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.

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.

Proprietary notice

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 https://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.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in the Arm documents.

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
1.0-F	Oct 6, 2020	Non-Confidential	Arm BBR 1.0
2.0-G	April 15, 2022	Non-Confidential	Arm BBR 2.0
2.1-H	April 15, 2024	Non-Confidential	Arm BBR 2.1
2.2-I	May 23, 2025	Non-Confidential	Arm BBR 2.2

Change history

The Change history tables describe the technical changes between released issues of this document in reverse order. Issue numbers match the revision history in Document release information.

Table 1: Differences between issue H and issue I

Change
• Arm BBR version 2.2 • Clarified CPPC requirements • Cleaned up CPPC wording • Clarified PSCI CPU Idle state • Updated with errata in Section 3.2 • Clarified PSCI and UEFI Reset mapping • Clarified the two different UARTs (Console and debug) for OS use • Updated references to all Arm ACPI tables • Updated SPCR and DBG2 to be conditionally required • Updated to SPCR v4 and default UART frequency • Provided reference to SMBIOS Type 44 structure • Provided requirements for system hibernate (S4) and runtime device Dx states • Provided requirements to ACPI power source and battery • Updated the reference of SMBIOS specification to version 3.8.0 • Updated the reference of UEFI specification to version 2.11 • Updated the reference of CXL specification to version 3.2 • Clarified SPCR console device must be in ACPI namespace • Updated FADT Preferred PM profile field for PCs • Mandated CPC EPP Register width at 8 bits • Added guidance and new fields in SMBIOS Type04 Processor Information Table • Updated the reference of ACPI specification to version 6.6

Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: https://developer.arm.com/glossary.

This document uses the following terms and abbreviations.

Terms and abbreviations

Term	Meaning
A64	The 64-bit Arm instruction set used in AArch64 state. All A64 instructions are 32 bits.

Term	Meaning
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.

Typographical conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`

Convention	Use
SMALL CAPITALS	Terms that have specific technical meanings as defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
	We recommend the following. If you do not follow these recommendations your system might not work.
	Your system requires the following. If you do not follow these requirements your system will not work.
	You are at risk of causing permanent damage to your system or your equipment, or of harming yourself.
	This information is important and needs your attention.
	This information might help you perform a task in an easier, better, or faster way.
	This information reminds you of something important relating to the current content.

Useful resources

This document contains information that is specific to this product. See the following resources for other relevant information.

- Arm Non-Confidential documents are available on developer.arm.com/documentation. Each document link in the tables below provides direct access to the online version of the document.
- Arm Confidential documents are available to licensees only through the product package.

Arm architecture and specifications	Document ID
Arm Architecture Reference Manual for A-profile architecture	DDI0487
Power State Coordination Interface (PSCI)	DEN0022
SMC Calling Convention (SMCCC)	DEN0028
Arm Functional Fixed Hardware (FFH) Specification	DEN0048
IO Remapping Table	DEN0049
Software Delegated Exception Interface (SDEI)	DEN0054
ACPI for Arm MPAM	DEN0065
ACPI for CoreSight	DEN0067
ACPI for CoreSight Performance Monitoring Unit Architecture	DEN0117
ACPI for Arm Components	DEN0093
Arm Base System Architecture	DEN0094
Arm Server Base System Architecture	DEN0029
Arm PC Base System Architecture	DEN0151
Arm True Random Number Generator Firmware Interface	DEN0098
ACPI for the Armv8-A RAS Extension and RAS System Architecture	DEN0085

Non-Arm resources	Organization
Advanced Configuration and Power Interface Specification Revision 6.6	UEFI Forum
Coherent Device Attribute Table (CDAT) Specification Version 1.03	UEFI Forum https://uefi.org/acpi
Compute Express Link (CXL) Specification Version 3.2	CXL Consortium
Redfish Host Interface Specification Version 1.3.0	DMTF
_DSD (Device Specific Data) Implementation Guide	UEFI Forum
Devicetree Specification 0.3	Devicetree.org
Embedded Base Boot Requirements	Arm
Firmware Windows Engineering Guide	Microsoft

Non-Arm resources	Organization
Intelligent Platform Management Interface 2.0, Revision 1.1 (October 2013)	Dell,HP,Intel,NEC
OCP Open System Firmware Checklist v1.1	OCP
PCI Firmware Specification Revision 3.3, and published ECNs and Errata, e.g., TPH ECN	PCI SIG
SMBIOS Reference Specification Version 3.8.0	DMTF
Unified Extensible Firmware Interface Specification. Version 2.11	UEFI Forum