Arm® Base Boot Requirements 1.0

Platform Design Document

Document number: DEN0044F

Arm Base Boot Requirements

Release information

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

Table 1-1 Change History

Date	Issue	Confidentiality	Change
6 Oct 2020	F	Non-Confidential	Arm BBR version 1.0

Arm document reference: LES-PRE-21585 version 4.0

1 About This Document

1.1 References

This document refers to the following documents:

Reference	Doc No	Authors	Title
[1]	ACPI 6.3	UEFI.org	Advanced Configuration and Power Interface Specification. Revision 6.3
[2]	Arm DDI 0487	Arm	Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile
[3]	Arm DEN 0006	Arm	Arm® TBBR Specification
[4]	Arm DEN 022	Arm	Power State Coordination Interface (PSCI) Version 1.1
[5]	Arm DEN 028	Arm	SMC Calling Convention (SMCCC) Version 1.2
[6]	Arm DEN 0054	Arm	Software Delegated Exception Interface (SDEI)
[7]	Arm DEN 056	Arm	System Control and Management Interface (SCMI) Version 2.0
[8]	Arm DEN 0085	Arm	ACPI for the Armv8 RAS Extensions 1.0
[9]	Arm DEN 0093	Arm	ACPI for Arm Components Version 1.0
[10]	Arm DEN 0094A	Arm	Arm® Base System Architecture Version 1.0
[11]	DMTF DSP0270	DMTF	Redfish Host Interface Specification Version 1.2.0
[12]	DT	Devicetree.org	Devicetree Specification 0.3
[13]	EBBR	Arm	Embedded Base Boot Requirements 1.01
[14]	IPMI	Dell, HP, Intel, NEC	Intelligent Platform Management Interface 2.0, Revision 1.1 (October 2013)
[15]	NIST800-147B	NIST	BIOS Protection Guidelines for Servers
[16]	NIST800-155	NIST	BIOS Integrity Measurement Guidelines
[17]	NIST800-193	NIST	Platform Firmware Resiliency Guidelines

[18]	OCP OSF Checklist	OCP	OCP Open System Firmware Checklist v1.0
[19]	PCI FW	PCI SIG	PCI Firmware Specification Revision 3.2
[20]	SMBIOS Version 3.4.0	DMTF	System Management BIOS (SMBIOS) Reference Specification
[21]	TCG ACPI	TCG	TCG ACPI Specification for TPM Family 1.2 and 2.0
[22]	TCG EFI	TCG	TCG EFI Protocol Specification, Family 2.0
[23]	TCG PC Firmware Profile v2.0	TCG	TCG PC Client Platform Firmware Profile Specification Family 2.0
[24]	TCG PPI	TCG	TCG PC Client Platform Physical Presence Interface Specification Family 1.2 and 2.0
[25]	TCG PTP	TCG	TCG PC Client Platform TPM Profile (PTP) Specification Family 2.0
[26]	TCG RAM	TCG	TCG Platform Reset Attack Mitigation Specification Version 1.00
[27]	UEFI Specification 2.8	UEFI.org	Unified Extensible Firmware Interface Specification. Version 2.8

1.1.1 Cross References

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

Examples:

ACPI § 5.6.5	-	Reference to the ACPI specification [1] section 5.6.6
UEFI § 6.1	-	Reference to the UEFI specification [27] section 6.1

1.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
A64	The 64-bit Arm instruction set used in AArch64 state. All A64 instructions are 32 bits.
AArch64 state	The Arm 64-bit Execution state that uses 64-bit general-purpose registers, and a 64-bit Program Counter (PC), Stack Pointer (SP), and Exception Link Registers (ELR). AArch64 Execution state provides a single instruction set, A64.

ACPI	Advanced Configuration and Power Interface.
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.
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
TCG	Trusted Computing Group
TPM	Trusted Platform Module
UEFI	Unified Extensible Firmware Interface.
UEFI Boot Services	Functionality that is provided to UEFI Loaded Images during the UEFI boot process.
UEFI Runtime Services	Functionality that is provided to an operating system after the ExitBootServices() call.

1.3 Feedback

Arm welcomes feedback on its documentation.

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

• The title Arm Base Boot Requirements.

• The document ID and version (DEN0044F 1.0).

• The page numbers to which your comments apply.

• A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

2 Background

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

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

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

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

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

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

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

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

3 Introduction

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

The primary goal of this document is to ensure sufficient standard system architecture to enable a suitably built single OS image to run on all hardware that is compliant with this specification and the Arm Base System Architecture specification[10]. A driver-based model for advanced platform capabilities beyond basic system configuration and boot is required. However, this model is beyond the scope of this document. Fully discoverable and describable peripherals aid the implementation of this type of a driver model.

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

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

This document is structured as following:

Section 4 BBR recipes

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

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

Section 5 PSCI/SMCCC requirements

Section 6 Secondary Core Boot requirements

Section 7 UEFI requirements

Section 8 ACPI requirements

Section 9 SMBIOS requirements

Section 10 ESBBR exceptions

Section 11 DeviceTree (DT) requirements

Appendixes

Provide summaries of required and recommended UEFI and ACPI interfaces.

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

4 BBR Recipes

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

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

Note: Servers that implement SBBR or LBBR recipes can be compliant with OCP Open System Firmware requirements by following the OCP OSF Checklist requirements [18].

4.1 SBBR recipe

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

SBBR-compliant systems must not present a DeviceTree binary to the operating system.

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

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

4.2 ESBBR recipe

Systems using ESBBR recipe must meet the requirements that are specified in section 4.1, with the exceptions that are specified in section 10.

4.3 EBBR recipe

Systems using EBBR recipe must meet the requirements that are specified in section 5 (PSCI/SMCCC) and in the EBBR specification v1.0.1[13].

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

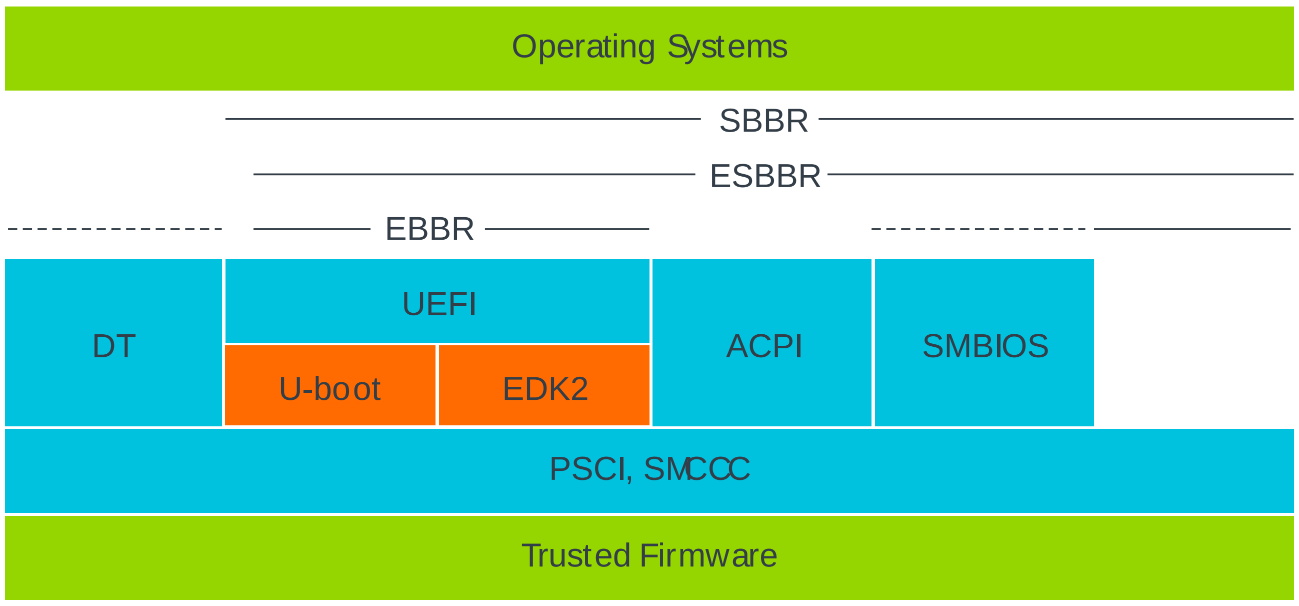
Note: Currently, Fedora, Debian, openSUSE, SLES, and Ubuntu support EBBR. This list of operating systems and hypervisors is subject to change.

4.4 LBBR recipe

LBBR is a recipe for LinuxBoot based systems. LinuxBoot is an alternative firmware stack that uses the Linux kernel as the Normal world firmware component. LinuxBoot is not a standard, but it can be supported on platforms that provide open source firmware and implement the requirements in section 5 (PSCI/SMCC), section 9 (SMBIOS), and section 8 (ACPI).

Note: LinuxBoot is used by some hyperscale datacenters that are compliant with the Open Compute Project (OCP) Open System Firmware (OSF) Checklist [18].

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



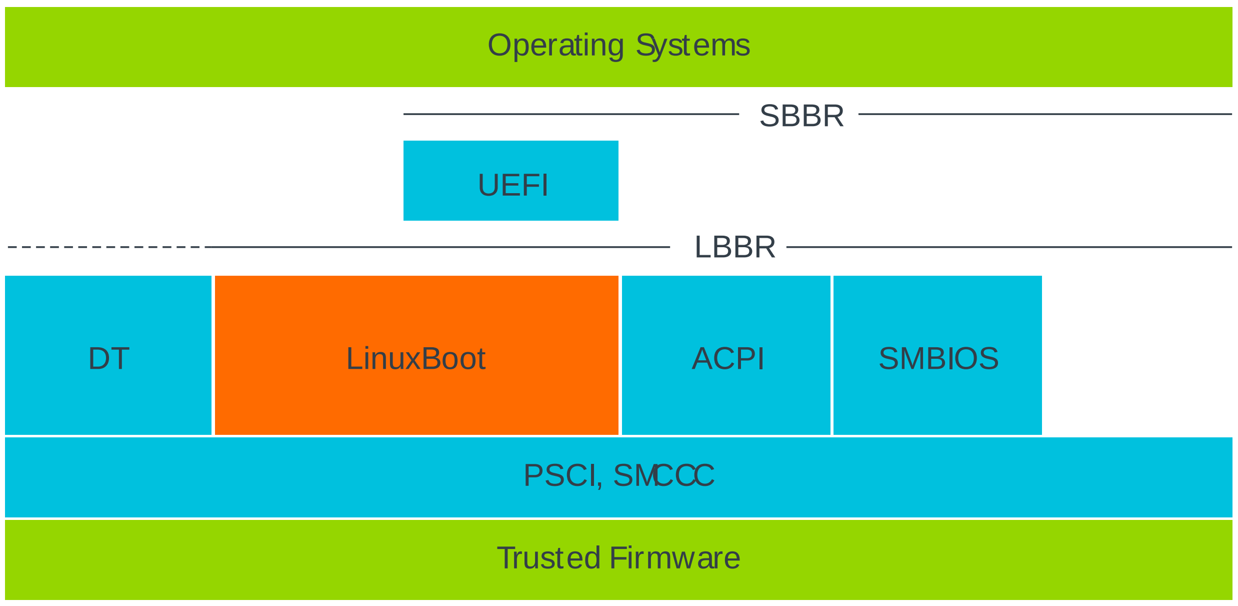


Figure 4-1 Relationships among SBBR, ESBBR, EBBR, and LBBR

5 PSCI/SMCCC Requirements

The system must expose PSCI[4] and SMCCC[5] interfaces to the operating system or hypervisor.

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

5.1 SMCCC Architecture Call requirements

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

5.2 PSCI Call requirements

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

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

6 Secondary Core Boot

Platforms providing EL3 must implement the Power State Coordination Interface (PSCI)[4]:

PSCI 1.1	Published April 2017

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

ACPI tables need to reflect:

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

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

All secondary cores remain powered down during boot. After boot, OSPM can call CPU_ON() into the PSCI firmware to power up a chosen core. The PSCI firmware powers up, initializes the core, and starts execution at the provided address.

7 UEFI Requirements

7.1 UEFI version

This document references the following specification and versions:

UEFI 2.8[27]	Published March 2019, includes the AArch64 bindings

7.2 UEFI compliance

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

7.3 UEFI system environment and configuration

7.3.1 AArch64 Exception levels

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

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

7.3.1.1 UEFI Boot at EL2

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

7.3.1.2 UEFI Boot at EL1

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

7.3.2 System volume format

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

7.3.3 UEFI image format

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

7.3.3.1 UEFI drivers

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

7.3.3.2 UEFI applications

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

7.3.3.3 PE/COFF image

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

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

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

7.3.4 GOP protocol

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

7.3.5 Address translation support

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

7.4 UEFI boot services

7.4.1 Memory map

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

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

The default RAM allocated attribute must be `EFI_MEMORY_WB`.

7.4.2 UEFI loaded images

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

7.4.3 Configuration tables

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

- `EFI_ACPI_20_TABLE_GUID`
  - The ACPI tables must be at version ACPI 6.3 or later with a HW-Reduced ACPI model. See section 8.
- `SMBIOS3_TABLE_GUID`
  - This table defines the 64-bit entry point for SMBIOS table.
  - The SMBIOS tables must conform to version 3.2.0 or later of the SMBIOS Specification. See section 9.

7.5 UEFI Runtime Services

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

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

7.5.1 Runtime Exception level

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

7.5.2 Runtime memory map

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

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

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

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

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

7.5.3 Real-time clock

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

- GetTime()
- SetTime()

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

7.5.4 UEFI reset and shutdown

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

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

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

- EfiWarmReset

These Runtime Services must be implemented by calling 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)
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 may call either interface to reset the system. Calling the UpdateCapsule() service requires the use of the UEFI Runtime service for reset.

7.5.5 Set variable

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

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

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

8 ACPI Requirements

8.1 ACPI version

This document references the following specification and versions:

ACPI 6.3[1]	Published January 2019, includes the Reduced HW profile

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

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

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

8.2 ACPI provided data structures

All platforms that comply with this specification must:

- Conform to the ACPI specification[1], version 6.3 or later
    - Legacy tables and methods are not supported.
- Implement the HW-Reduced ACPI model. See ACPI § 3.11.1 and 4.1.
- Not support legacy ACPI Fixed Hardware interfaces
- Provide either Interrupt-signaled Events (see ACPI § 5.6.9) or GPIO-signaled Events (see ACPI § 5.6.5) for the conveyance of runtime event notifications, from the system firmware to the Operating System Power Management (OSPM).

8.3 ACPI tables

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

All platforms that comply with this specification:

- Must ensure that the structure of all tables is consistent with the ACPI 6.3 or later specification
    - Legacy tables are not supported.
- Must ensure that the pointer to the RSDP is passed through UEFI to the OSPM as described by UEFI
- Must use 64-bit addresses within all address fields in ACPI tables
    - This restriction ensures a long-term future for the ACPI tables. Versions before ACPI 5.0 allowed 32-bit address fields.

8.3.1 Mandatory ACPI tables

The following tables are mandatory for all compliant systems.

8.3.1.1 RSDP

- Root System Description Pointer (RSDP), ACPI § 5.2.5.

- Within the RSDP, the RsdtAddress field must be null (zero) and the XsdtAddresss must be a valid, non-null, 64-bit value.

8.3.1.2 XSDT

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

8.3.1.3 FADT

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

8.3.1.4 DSDT and SSDT

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

8.3.1.5 MADT

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

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

8.3.1.6 GTDT

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

8.3.1.7 DBG2

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

8.3.1.8 SPCR

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

8.3.1.9 MCFG

- PCI Memory-mapped Configuration Space (MCFG). PCI FW[19] § 4.1.2
    - This table described the PCIe ECAM base address
    - This table is required if PCIe is supported.

8.3.1.10 PPTT

- Processor Properties Topology Table (PPTT), ACPI § 5.2.29.
    - This table describes the topological structure of processors that are controlled by the OSPM and their shared resources.

8.3.2 Recommended ACPI tables

ACPI tables that are recommended are listed in Appendix E .

Not every platform that is compliant with this specification provides all of these tables. This is because many tables reference optional platform features.

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.

8.3.3 Optional ACPI tables

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

8.4 ACPI definition blocks

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

8.5 ACPI methods and objects

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

All objects and methods must conform to the definitions in ACPI version 6.3 or later. Legacy definitions are not supported.

8.5.1 Global methods and objects

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

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

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

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

8.5.2 Device methods and objects

For each device definition in the platform DSDT or SSDT tables, platforms must provide the following predefined methods or objects, in accordance with their definitions in version 6.3 or later of the ACPI specification:

- _ADR: Address on the parent bus of the device. Either this object or the _HID must be provided. This object is essential for PCI, for example.
    - See ACPI § 6.1.1
- _CCA: Cache Coherency Attribute. This object provides information about whether a device has to manage cache coherency and about hardware support. This object is mandatory for all devices that are not cache-coherent, and recommended for all devices. This object is only relevant for devices that can access CPU-visible memory, like devices that are DMA capable.
    - See ACPI § 6.2.17
- _CRS: Current Resource Settings. This method provides essential information to describe resources, like registers and their locations, that are provided by the device.
    - See ACPI § 6.2.2.

    - **Note:** The _PRS (Possible Resource Settings) and _SRS (Set Resource Settings) are not supported.
- _HID: Hardware ID. This object provides the Plug and Play Identifier or the ACPI ID for the device. Either this object or the _ADR must be provided.
    - See ACPI § 6.1.5.
- _STA: Status. This method identifies whether the device is on, off, or removed.
    - See ACPI § 6.3.7 and 7.2.4.
- _UID: Unique persistent ID. This object provides a unique value that is persistent across boots, and can uniquely identify the device with either a common _HID or _CID. The object is used, for example, to identify a PCI root bridge, if there are multiple PCI root bridges in the system.
    - See ACPI § 6.1.12.

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

8.5.3 GPIO controllers

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

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

8.5.4 Generic Event Devices

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

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

8.5.5 Address translation support

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

8.5.6 Describing Arm components in ACPI

Certain components that are implemented or licensed by Arm have special ACPI properties and IDs defined in ACPI for Arm Components [9]. If a platform includes any of the Arm components that are listed in ACPI for Arm Components, it must follow the guidance of that specification for implementing the ACPI objects and methods for such components.

8.6 Hardware requirements imposed on the platform by ACPI

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

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

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

- For the ACPI Platform Error Interface (APEI):
    - One event for non-fatal error signaling (ACPI § 18.3.2.7.2)
    - Software Delegated Exception(SDE)[6] or one NMI-equivalent signal for use in fatal errors
    - See ACPI § 18.
- At least one wake signal, which is routed through a platform event.

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

8.6.1 Processor Performance Control

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

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

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

8.6.2 Time and Alarm Device

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

9 SMBIOS Requirements

The System Management BIOS (SMBIOS) [20] that is published by the DMTF is an important firmware component for servers. SMBIOS provides basic hardware and firmware configuration information through table-driven data structures. Although it is not required for operating system booting or core kernel functions, SMBIOS is widely used for platform management, scripting, and deployment applications.

9.1 SMBIOS version

This document references the following specification and versions:

SMBIOS 3.4.0 Published September 2020.

Legacy SMBIOS tables and formats are not supported.

9.1.1 SMBIOS requirements on UEFI

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

9.2 SMBIOS structures

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

9.2.1 Type00: BIOS Information (required)

- Vendor
- BIOS Version
- BIOS Release Date
- BIOS ROM Size
- System BIOS Major Release
- System BIOS Minor Release
- Embedded Controller Firmware Major Release
- Embedded Controller Firmware Minor Release
- Extended BIOS ROM Size

9.2.2 Type01: System Information (required)

- Manufacturer
- Product Name
- Version
- Serial Number
- UUID
  - This field must provide a unique value for every individual system.
- SKU Number

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

- Manufacturer
- Product
- Version

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

9.2.4 Type03: System Enclosure or Chassis (required)

- Manufacturer
- Type
- Version
- Serial Number
- Asset Tag Number
- Height
- SKU Number

9.2.5 Type04: Processor Information (required)

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

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

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

9.2.6 Type07: Cache Information (required)

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

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

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

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

- Slot Designation
- Slot Type
- Slot Data Bus Width
- Current Usage
- Slot ID
- Slot Characteristics 1
- Slot Characteristics 2
- Segment Group Number
- Bus Number
- Device Function Number

9.2.9 Type11: OEM Strings (recommended)

- Count

9.2.10 Type13: BIOS Language Information (recommended)

- Installable Languages
- Flags
- Current Language

9.2.11 Type15: System Event Log (recommended)

9.2.12 Type16: Physical Memory Array (required)

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

9.2.13 Type17: Memory Device (required)

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

- Extended Size
- Extended Speed

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

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

9.2.14 Type19: Memory Array Mapped Address (required)

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

9.2.15 Type32: System Boot Information (required)

- Boot Status

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

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

Note: The ACPI SPMI Table replaces this Type in IPMI v1.5 and v2.0 [14]

9.2.17 Type41: Onboard Devices Extended Information (recommended)

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

9.2.18 Type42: Redfish Host Interface (required for platforms supporting Redfish Host Interface[11])

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

10 ESBBR Exceptions

ESBBR recipe (4.2) allows exceptions (as defined in this section) to the SBBR recipe (4.1).

Note: There are no exceptions defined in v1.0 of this document.

11 DT Requirements

The Devicetree Specification [12] specifies a construct called a DeviceTree to describe system hardware. DeviceTree describes device information in a system that cannot necessarily be dynamically detected.

11.1 DT version

This document references the following specification and versions:

Devicetree Specification Release 0.3	Published Feb 2020

11.2 DT requirements

Chapter 3 of the Devicetree Specification documents the requirements for the base set of device nodes that are required in a DT-compliant devicetree.

Appendix A Required UEFI Boot Services

Service	UEFI §
EFI_RAISE_TPL	7.1
EFI_RESTORE_TPL	7.1
EFI_ALLOCATE_PAGES	7.2
EFI_FREE_PAGES	7.2
EFI_GET_MEMORY_MAP	7.2
EFI_ALLOCATE_POOL	7.2
EFI_FREE_POOL	7.2
EFI_CREATE_EVENT	7.1
EFI_SET_TIMER	7.1
EFI_WAIT_FOR_EVENT	7.1
EFI_SIGNAL_EVENT	7.1
EFI_CLOSE_EVENT	7.1
EFI_INSTALL_PROTOCOL_INTERFACE	7.3
EFI_REINSTALL_PROTOCOL_INTERFACE	7.3
EFI_UNINSTALL_PROTOCOL_INTERFACE	7.3
EFI_HANDLE_PROTOCOL	7.3
EFI_REGISTER_PROTOCOL_NOTIFY	7.3
EFI_LOCATE_HANDLE	7.3
EFI_LOCATE_PROTOCOL	7.3
EFI_LOCATE_DEVICE_PATH	7.3
EFI_INSTALL_CONFIGURATION_TABLE	7.5
EFI_LOAD_IMAGE	7.4
EFI_START_IMAGE	7.4
EFI_EXIT	7.4
EFI_UNLOAD_IMAGE	7.4
EFI_EXIT_BOOT_SERVICES	7.4
EFI_GET_NEXT_MONOTONIC_COUNT	7.5
EFI_STALL	7.5
EFI_SET_WATCHDOG_TIMER	7.5
EFI_CONNECT_CONTROLLER	7.3
EFI_DISCONNECT_CONTROLLER	7.3
EFI_OPEN_PROTOCOL	7.3
EFI_CLOSE_PROTOCOL	7.3
EFI_OPEN_PROTOCOL_INFORMATION	7.3
EFI_PROTOCOLS_PER_HANDLE	7.3
EFI_LOCATE_HANDLE_BUFFER	7.3
EFI_LOCATE_PROTOCOL	7.3
EFI_INSTALL_MULTIPLE_PROTOCOL_INTERFACES	7.3
EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES	7.3
EFI_CALCULATE_CRC32	7.5
EFI_COPY_MEM	7.5
EFI_SET_MEM	7.5
EFI_CREATE_EVENT_EX	7.1

Appendix B Required UEFI Runtime Services

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

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

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

Note: If the system implements UEFI capsule services, OSes need to clean cache by VA to PoC (that is, using DC CVAC) on each ScatterGatherList element that is passed to the UpdateCapsule() before issuing the call. This is needed only when UpdateCapsule() is called after ExitBootServices(). This requirement will be clarified in a future version of the UEFI specification.

UEFI Configuration Table Entries

Configuration Table
EFI_ACPI_20_TABLE_GUID
SMBIOS3_TABLE_GUID

Appendix C Required UEFI Protocols

Core UEFI Protocols

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

Media I/O Protocols

Service	UEFI §
EFI_LOAD_FILE_PROTOCOL	13.1
EFI_LOAD_FILE2_PROTOCOL	13.2
EFI_SIMPLE_FILE_SYSTEM_PROTOCOL	13.4
EFI_FILE_PROTOCOL	13.5

The Load File protocol is used to obtain files from arbitrary devices that are primarily boot options. The Load File 2 protocol is used to obtain files from arbitrary devices that are not boot options.

Console Protocols

Service	UEFI §
EFI_SIMPLE_TEXT_INPUT_PROTOCOL	12.3
EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL	12.2
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL	12.4

Driver Configuration Protocols

Service	UEFI §
EFI_HII_DATABASE_PROTOCOL	34.8
EFI_HII_STRING_PROTOCOL	34.3
EFI_HII_CONFIG_ROUTING_PROTOCOL	35.4
EFI_HII_CONFIG_ACCESS_PROTOCOL	35.5

Random Number Generator Protocol

Service	**UEFI §**
EFI_RNG_PROTOCOL	37.5

The EFI_RNG_PROTOCOL is used by operating systems for entropy generation capabilities early during OS boot. The protocol is recommended to support returning at least 256 bits of full entropy in a single call, from a source with security strength of at least 256 bits.

Appendix D Optional UEFI Protocols

Basic Networking Support

Service	UEFI §
EFI_SIMPLE_NETWORK_PROTOCOL	24.1
EFI_MANAGED_NETWORK_PROTOCOL	25.1
EFI_MANAGED_NETWORK_SERVICE_BINDING_PROTOCOL	25.1

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

Network Boot Protocols

Service	UEFI §
EFI_PXE_BASE_CODE_PROTOCOL	24.3
EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL	24.4
EFI_MTFTP4_PROTOCOL	30.3
EFI_MTFTP6_PROTOCOL	30.4

Ipv4 Network Support

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

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

Ipv6 Networking Support

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

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

VLAN Protocols

Service	UEFI §
EFI_VLAN_CONFIG_PROTOCOL	27.1

iSCSI Protocols

Service	UEFI §
EFI_ISCSI_INITIATOR_NAME_PROTOCOL	16.2

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

REST Protocols

Service	UEFI §
EFI_REST_EX_PROTOCOL	29.7.2.2
EFI_REST_EX_SERVICE_BINDING_PROTOCOL	29.7.2.1
EFI_REDFISH_DISCOVER_PROTOCOL	31.1.4
EFI_REST_JSON_STRUCTURE_PROTOCOL	29.7.3.2

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

HTTP Network Protocols

Service	UEFI §

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

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

Firmware Update

Service	**UEFI §**
EFI_FIRMWARE_MANAGEMENT_PROTOCOL	23.1
EFI_SYSTEM_RESOURCE_TABLE GUID (ESRT) (UEFI Configuration Table)	23.4

Note: Firmware Management Protocol is used for device firmware update if the device UEFI drivers support it.

Note: The ESRT configuration table is used for firmware update If the system implements UEFI capsule services.

Appendix E Recommended Acpi Tables

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

Platform Error Interfaces

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

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

NUMA

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

ACPI Signature	Full Name	ACPI §
SLIT	System Locality Information Table	5.2.17
SRAT	System Resource Affinity Table	5.2.16
HMAT	Heterogeneous Memory Attribute Table	5.2.27

Platform Communications Channel (PCC)

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

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

Platform Debug Trigger

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

ACPI Signature	**Full Name**	**ACPI §**
PDTT	Platform Debug Trigger Table	5.2.28

NVDIMM Firmware Interface

NFIT describes NVDIMM that is required if NVDIMM is supported.

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

Graphics Resource Table

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

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

IPMI

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

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

Appendix F Recommended ACPI Methods

CPU performance control

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

Method	Full Name	ACPI §
_CPC	Continuous Performance Control (replaces _PCT and _PSS)	8.4.6.1
_PSS	Performance Supported States (Superseded by _CPC)	8.4.6.2

CPU and system idle control

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

Method	Full Name	ACPI §
_LPI	Low Power Idle States	8.4.4

NUMA

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

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

IPMI

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

Method	Full Name	ACPI §
_IFT	IPMIv2: the IPMI Interface Type, if and only if IPMI has been implemented	5.5.2.4.4, IPMI spec [14]
_SRV	IPMIv2: the IPMI revision that is supported by the platform, if and only if IPMI has been implemented	IPMI spec [14]

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.	6.2.5
_DSM	Device Specific Method (used to convey info to ACPI that it might not currently have a mechanism to describe, see https://lkml.org/lkml/2013/8/20/556)	9.1.1
_INI	Initialize a device	6.5.1

Resources

The following methods provide descriptions for devices.

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