Illustrative OS-A-SEE Specification

v7, Andrei Warkentin, Intel Corporation

This document is heavily inspired by the Arm (R) DEN0044G Base Boot Requirements 2.0 Platform Design Document, and is meant to fuel further discourse. The reason why the BBR is taken as a base is because to date it represents the single authoritative effort to describe the booting requirements congruent with industry norms for building servers and PC-like systems, and because it additionally accommodates requirements for building embedded-style systems. There's a lot of trial-by-fire knowledge accumulated and refined there, that we should build on. Also takes into account:

- RISC-V ACPI Guidance document
  - riscv-acpi/riscv-acpi-guidance.adoc at master · riscv-non-isa/riscv-acpi · GitHub
- EBBR specification
  - https://arm-software.github.io/ebbr/

Parts of the document with **XXX** signify gaps in specifications that need to be addressed. Notable gaps:

- Narrative text should/need to/ought to be rewritten instead of being (in many places) taken verbatim from the BBR doc. The “SANITIZED UP TO” comment will be used as a marker of progress.
  UEFI calling conventions for RISC-V need to be fixed to not be so restrictive about how an implementation works and what mode it starts in (**https://bugzilla.tianocore.org/show_bug.cgi?id=4232**)
- ACPI RISC-V FFH spec (**XXX: link to WIP work**)
  - E.g. for LPI, CPPC
  - For SBI ecall-backed OperationRegions
- Arm SDEI-like spec, or similar NMI-like mechanism (needed for signaling fatal APEI errors)
- APEI RISC-V definitions (CPER)
- ACPI table for IOMMU (obviously dependent on IOMMU). This is not a blocker - IOMMU-related content can be hidden until a revision of the spec that appears once the IOMMU side of things is ratified (**https://github.com/riscv-non-isa/riscv-acpi/blob/master/rimt.adoc**)
- Need a separate RVI doc to provide a “solar-system-at-a-glance” view of how HW/FW specs compose towards a solution.
- Need more “official” review from various OSVs and participation in the OS-A-SEE.

Areas of difference from the BBR:

- Covers (or doesn't preclude) virtual machine environments
- Doesn't include the LBBR-like recipes
- Fleshes out UEFI required / optional boot protocols
  - This list is meant to cover protocols used by known OS first stage loaders

    - The BBR list is missing block I/O protocols, PCIe protocols
- Adds references to more ACPI tables: NBFT alongside IBFT, TPM2

Changelog	
Version	**Changes**
v7	Clarified MADT entries (can have IMSICs only, IMSICs + APLICs, 1 APLIC or 1 PLIC)
v6	Applies feedback from Canonical ● Make chapter 8 relevant to VI, but don't provide overlapping requirements for VI. ● Adjusted description of UEFI memory map ● Make DBG2 optional ● Reworded use of VA (virtual address) in text around capsule update. Fixed some formatting/missing text issues.
v5	Applies feedback from Ventana. ● Reference the WIP Type44 SMBIOS structure. ● References UEFI Boot Protocol spec. ● APLIC is optional. ● GICC → RINTC Applies feedback from Canonical. ● Reference SBI 1.0 spec. ● Require SBI HSM for secondary core boot. ● Clarify that ACPI is mandatory only for HI. DTB is mandatory for VI. Clarify that exposing DT and ACPI at the same time is disallowed. ● Mark SMBIOS Type03 as optional. ● Refine definition of socket for Type04 to include certain chiplet scenarios ● Move ESRT under conditionally-required tables. Applies feedback from VMware ● TAD language to allow operation if a dependency is not present (e.g. OS driver for I2C/SPI bus)

v4	Applies feedback received from VMware.
	
	• Replace IORT references with **XXX**
	• Clarify use of RT service for system reset over SBI, instead of allowing either to be used. Allowing either to be used is a compromise that had to be made in SystemReady to accomodate existing OS behavior. Consequently relax the UEFI RT implementation to allow non-SBI based implementations.
	• Because of clarifications around use of EFI ResetSystem(), relax language around how variable services could be implemented, which allows persistent variable support even on systems where these could be stored on OS-visible secondary storage.
	• Clarify CPU Performance Control to only rely on _CPC (as per CPPC)
	• Clarify how RTC is exposed
	  ○ If RTC is on an OS-managed bus (e.g. I2C, where the OS can access the bus for any reason, such as other devices), the RTC must be exposed via TAD.
	  ○ If the RTC is not on an OS-managed bus (i.e. not subject to any arbitration issues between concurrent OS and UEFI accesses to the underlying hardware) the RTC must be exposed via UEFI RT services.
	• Add references to EFI_CONFORMANCE_TABLE and EFI_RT_PROPERTIES_TABLE

v1/v2/v3	Original with minor typo fixes, add conformance profile info.

1. About this Document

1.1. References

This document refers to the following documents:

Reference	Doc #	Authors	Title
[1]	ACPI	UEFI.org	Advanced Configuration and Power Interface Specification. Revision 6.6.
[2]	v1.0.0	RVI	RISC-V Supervisor Binary Interface Specification
[3]	v1.0.0	RVI	RISC-V UEFI Protocol Specification
[13]	v1.3	TCG	TCG ACPI Specification
[14]	CDAT	UEFI.org	Coherent Device Attribute Table (CDAT) Specification

[15]	CXL	CXL Consortium	Compute Express Link (CXL) Specification 2.0, and published ECNs and Errata
[16]	DMTF DSP0270	DMTF	Redfish Host Interface Specification Version 1.2.0
[17]	_DSD Implementation Guide	UEFI.org	_DSD (Device Specific Data) Implementation Guide
[18]	DT	Devicetree.org	Devicetree Specification 0.3
[19]	EBBR	Arm	Embedded Base Boot Requirements 2.0.1
[20]	WEG SMBIOS	Microsoft	Firmware Windows Engineering Guide
[21]	IPMI	Dell, HP, Intel, NEC	Intelligent Platform Management Interface 2.0, Revision 1.1 (October 2013)
[22]	OCP OSF Checklist	OCP	OCP Open System Firmware Checklist v1.0
[23]	PCI FW	PCI SIG	PCI Firmware Specification Revision 3.3, and published ECNs and Errata, e.g., TPH ECN
[24]	SMBIOS	DMTF	System Management BIOS (SMBIOS) Reference Specification Version **XXX**
[25]	UEFI	UEFI.org	Unified Extensible Firmware Interface Specification. Version 2.9

1.1.1. Cross References

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

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

1.2. Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
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 UEFI Boot Services and UEFI Runtime Services
Hart	Hardware thread in RISC-V. This is the hardware execution context that contains all the state mandated by the ISA.
HS-Mode	Hypervisor-extended-supervisor mode which virtualization the supervisor mode
HSM	Hart State Management (HSM) is an SBI extension, that enables the supervisor mode software to implement ordered booting
HDM	Host-managed Device Memory, such as CXL attached memory
M-Mode	Machine-mode is the most secure and privileged mode in RISC-V
OEM	Original Equipment Manufacturer. In this document, the final device manufacturer.
RISC-V	An open standard Instruction Set Architecture (ISA) based on Reduced Instruction Set Architecture (RISC)
RV64	64-bit execution mode in RISC-V
RVI	RISC-V International
SEE	Supervisor Execution Environment. Not to be confused with OS-A SEE, this is formed by components running in M-Mode or HS-Mode
SBI	RISC-V Supervisor Binary Interface. This is an interface between SEE and S-Mode software
OS-A SEE	OS-A Supervisor Execution Environment Specification
SiP	Silicon Partner. In this document, the silicon manufacturer
S-Mode	Supervisor-mode, reserved for OS kernels
VS-Mode	Virtualized supervisor mode where the guest OS is expected to run when hypervisor is enabled.
SMBIOS	System Management BIOS
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 UEFI Loaded Images during the UEFI boot process and to an operating system after the ExitBootServices() UEFI Boot Service call.

2. Background

RISC-V cores are used in a wide variety of 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 RISC-V architecture profiles segment RISC-V-based 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.

<insert here bit about RISC-V's extensibility>

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.

RISC-V cores 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 OS-A Supervisor Execution Environment (OS-A SEE) is part of the RVI strategy of addressing this variability.

3. Introduction

This document specifies the OS-A Supervisor Execution Environment (OS-A SEE) requirements for Boot and Runtime Services, that system software, for example operating systems and hypervisors, can rely on. This specification targets 64-bit RISC-V processors capable of running typical ("general purpose") operating systems and hypervisors.

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. 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 RVI and industry standard firmware interfaces applicable. They include the SBI, UEFI, ACPI, SMBIOS, and DT interfaces. Requirements that are based on these interfaces are specified. In addition, various recipes are created to accommodate the target market segments, products, operating systems and hypervisors.

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

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

4. Recipes

This section provides recipes to accommodate various operating systems and hypervisors. It is expected that these operating system and hypervisor requirements will be largely driven by target market segments. The HI (Horizontally Integrated) recipe is most appropriate for building solutions combining components (hardware, software, off-the-self) from multiple providers and with a focus on interoperability. Examples of such solutions include server and client devices. The VI (Vertically Integrated) recipe is appropriate for solutions where all facets (hardware, software) are driven end-to-end by a single provider. Examples of such solutions include embedded systems and cloud servers. The application of a recipe to a product/segment is a recommendation, not a requirement.

These recipes define the Boot and Runtime Services for a physical or virtual machine system, including services that are required for virtualization.

4.1. OS-A-SEE-HI (Horizontally-Integrated) Recipe

Systems using OS-A-SEE-HI recipe must meet the requirements that are specified in 5. RISC-V Processor Requirements, 6. SBI Requirements, 7. Secondary Core Boot, 8. UEFI Requirements, 9. ACPI Requirements, and 10. SMBIOS Requirements.

Currently, **XXX** operating systems require OS-A-SEE-HI recipe. Other operating systems, for example **XXX** can also support OS-A-SEE-HI. 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, OS-A-SEE-HI compliance does not require EDK2 implementation

4.2. OS-A-SEE-VI (Vertically-Integrated) Recipe

Systems using OS-A-SEE-VI recipe must meet the requirements that are specified in 5. RISC-V Processor Requirements, 6. SBI Requirements, 8. UEFI Requirements) and in the EBBR specification v2.0.1[19].

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

Note: Currently, **XXX** operating systems support OS-A-SEE-VI. This list of operating systems and hypervisors is subject to change.

5. RISC-V Processor Requirements

This specification is minimally proscriptive on the RISC-V processor requirements. It is anticipated that such detailed requirements will be largely driven by target market segment and product/solution requirements. The SEE mandates the minimal set required for a compliant implementation of the SEE alone.

The SEE requires processor cores to be at least compliant to:

- RVA20S64, as OS-A-SEE governs the interactions between 64-bit OS supervisor-mode software and 64-bit firmware.
- Sm1p11 or hypervisor extension, to sustain SEE components interfaced via SBI.

These are minimum requirements, allowing for the wide variety of existing and future processor implementations to be supported with OS-A-SEE. It is expected that operating systems and hypervisors may impose additional profile/ISA requirements, depending on the use-case and application.

6. SBI Requirements

The resident firmware in M mode or HS-mode must implement and conform to at least SBI [2] v1.0.0.

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

The following extensions are mandatory:

- HSM (Hart State Management)
- System Reset Extension

7. Secondary Core Boot

The SBI HSM extension is the only mechanism for booting secondary cores, implementing CPU idling, and providing reset and shutdown runtime services.

Where CPU idling low power states are provided, the DSDT must provide _LPI objects. Secondary cores can be brought online for specific tasks by calling **sbi_hart_start** into SBI. A secondary core brought online by firmware services before OS load must be put offline again by calling **sbi_hart_stop** into SBI to force the cores into OS compatible state before handing off control to OS.

If the cores are being manipulated internally to a UEFI implementation, this should happen before completing the EFI_EVENT_GROUP_READY_TO_BOOT event. After boot, an OS can call **sbi_hart_start** to start a chosen secondary core.

8. UEFI Requirements

8.1. UEFI Version

This document references the following specification and versions:

- UEFI 2.10 [25] Published August 2022, includes the RISC-V bindings.
- RISC-V UEFI PROTOCOL Specification, v1.0.0

8.2. 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, and in the RISC-V UEFI Protocol Specification [3].

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

8.3. UEFI System Environment and Configuration

The resident RISC-V UEFI boot-time environment is specified to use the highest 64-bit non-M-mode available. This level is either S-Mode, VS-Mode, or HS-Mode, depending on whether virtualization is used or supported.

8.3.1. Privilege Levels

Resident UEFI firmware might target a specific privilege level. In contrast, UEFI loaded image drivers and boot applications, must not contain any built-in assumptions of the privilege level at boot time. This is because these UEFI loaded images can be loaded into (V)S-Mode or HS-Mode.

As an implementation choice, UEFI firmware may start execution in M-Mode. However, it must switch to the highest non-M-mode implemented before starting up EFI drivers or applications.

8.3.1.1. UEFI boot in HS-Mode

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

8.3.1.2. UEFI boot in (V)S-Mode

Booting of UEFI at (V)S-Mode is only permitted in the following distinct conditions:

- on systems that do not implement HS-Mode.
- within a virtual machine 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.

8.3.2. 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]

8.3.3. UEFI Image Format

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

8.3.3.1. 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 RV64 binary format.

8.3.3.2. UEFI applications

A UEFI application [UEFI § 2.1.2] must be in the RV64 binary format to be used for the systems that comply with this specification.

8.3.3.3. 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. 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.

8.3.4. 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).

8.3.5. 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, or hot plug), or report no resources assigned.

8.4. UEFI Boot Services

8.4.1. 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 runtime services.
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.

8.4.2. UEFI loaded images

UEFI loaded images for RV64 must be in 64-bit PE/COFF format and must contain only RV64 code.

8.4.3. Configuration tables

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

- OS-A-SEE-HI Recipe
  - EFI_ACPI_20_TABLE_GUID
    - The ACPI tables must be at version ACPI 6.6 or later with a HW-Reduced ACPI model. See 9. ACPI Requirements.
  - SMBIOS3_TABLE_GUID
    - This table defines the 64-bit entry point for the SMBIOS tables. 2cfd2657-db2c-4ae6-a37e-93524c5db5ff
    - The SMBIOS tables must conform to version **XXX** or later of the SMBIOS Specification.
- EFI_CONFORMANCE_PROFILE_TABLE
  - The following GUID in the EFI Conformance Profile Table is used to indicate compliance to version 1.0.0 of this specification.
    - ```
      #define EFI_CONFORMANCE_PROFILE_OS_A_SEE_1_0_GUID
      { 0x05453310, 0x0545, 0x0545, \
      { 0x05, 0x45, 0x33, 0x05, 0x45, 0x33, 0x05, 0x45 }}
      ```
    - If the platform advertises the GUID in the EFI Conformance Profile Table, then it must be compliant with the corresponding version of this specification.

A compliant implementation must under no circumstances expose both ACPI and DTB tables at the same time.

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

- EFI_RT_PROPERTIES_TABLE_GUID
  - This optional table should be published by a platform, if it no longer supports all EFI runtime services once ExitBootServices() has been called by the OS.

8.5. UEFI Runtime Services
      ```

UEFI Runtime Services exist after the ExitBootServices() Boot Service 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.

8.5.1. Runtime Exception Level

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

8.5.2. 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.

8.5.3. Real-time Clock

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

- GetTime()
- SetTime()

If the RTC is not subject to any arbitration issues between concurrent OS and UEFI accesses to the underlying hardware, the RTC must be exposed via UEFI RT services.

For systems where the RTC is on an OS-managed bus (e.g. I2C, subject to arbitration issues due to access to the bus by the OS), the RTC must be implemented in terms of an ACPI Time and Alarm Device (TAD), to ensure all RTC accesses are done via the OS driver stack, and SetTime must return EFI_UNSUPPORTED when accessed after ExitBootServices().

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

If the SetTime() implementation returns EFI_UNSUPPORTED after ExitBootServices(), it must be appropriately described in an EFI_RT_PROPERTIES_TABLE.

8.5.4. 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 EfiResetWarm command.

These Runtime Services must be implemented using the applicable SBI services. The following table maps the UEFI and SBI reset calls:

EFI ResetSystem() ResetType	sbi_system_reset type
EfiResetShutdown	Shutdown
EfiResetCold	Cold reboot
EfiResetWarm	Warm reboot
EfiResetPlatformSpecific, with a platform-specific ResetData GUID	IMPLEMENTATION DEFINED.

Note: Operating systems must call UEFI Runtime Services to reset the system, preferring this to SBI or other platform-specific mechanisms. This allows for UEFI implementations to perform required platform tasks (e.g. servicing UpdateCapsule() calls, or persisting non-volatile variables in certain implementations).

8.5.5. Set Variable

Non-volatile UEFI variables must persist across EFI ResetSystem() calls.
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.

8.6. Firmware Update

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

If the firmware update is performed in-band (firmware on the application processor updates itself), then the firmware must implement either:

- UpdateCapsule() [UEFI § 8.5.3].

    - Firmware must accept updates in the "Firmware Management Protocol Data Capsule Structure" format as described in "Delivering Capsules Containing Updates to Firmware Management Protocol" [UEFI § 23.3].
- Delivery of Capsules via file on Mass Storage Device [UEFI § 8.5.5]

Firmware must also provide an ESRT [UEFI § 23.4] that describes every firmware image that is updated in-band.

For in-band firmware updates at runtime, the firmware must support processing
If the firmware update is performed out-of-band (for example, by an independent Baseboard Management Controller (BMC)), then the platform is not required to implement UpdateCapsule() or the alternative Capsule update via Mass Storage Device.

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

Note: If an OS uses UEFI capsule services, the OS must clean the data cache for the memory ranges corresponding to each ScatterGatherList element that is passed to the UpdateCapsule() before issuing the call. The cache clean operation used must be the strongest available in the platform. This is required only when UpdateCapsule() is called after ExitBootServices(). This requirement is clarified in UEFI specification [UEFI § 8.5.3].

9. ACPI Requirements

9.1. ACPI Version

This document references the following specification and versions:
ACPI 6.6[16] [1] Published **XXXX 202X**

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, the OS to drive the Operating System-directed Power Management (OSPM).

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

9.2. ACPI Provided Data Structures

All platforms that comply with this specification must:

- Conform to the ACPI specification[16], version 6.6 or later. Legacy tables and methods are not supported.
- Implement the HW-Reduced ACPI model. [ACPI § 3.11.1][ACPI § 4.1].
- Not support legacy ACPI Fixed Hardware interfaces.
- Provide either Interrupt-signaled Events [ACPI § 5.6.9] or GPIO-signaled Events [ACPI § 5.6.5] for the conveyance of runtime event notifications, from the system firmware to the Operating System Power Management (OSPM).
- Implement **XXX** RISC-V FFH specification (**XXX ...mapping to SBI calls, CSR accesses**)
- Implement **XXX** RISC-V specification for NMI/NMI-like firmware calls into the OS (**XXX similar to Arm SDEI ?**)

9.3. 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, flash or DDR memory.
All platforms that comply with this specification:

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

9.3.1. Mandatory ACPI Tables

The following tables are mandatory for all compliant systems.

9.3.1.1. 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.

9.3.1.2. 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 the other ACPI tables that the OSPM will use.

9.3.1.3. 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.
    - Select an appropriate Preferred PM Profile, as defined by the ACPI specification is selected [ACPI § 5.2.9.1].
        - For servers, select one of the server profiles.

9.3.1.4. 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.

9.3.1.5. MADT

- Multiple APIC Description Table (MADT) [ACPI § 5.2.12]
    - This table describes the harts
    - This table describes the interrupt controllers, their version, and their configuration.
        - On AIA systems with support for MSIs, the MADT must include an IMSIC entry for each hart in the system (and optionally, 1 or more APLIC entries)
        - On AIA systems without support for MSIs, the MADT must include 1 APLIC entry.
        - On PLIC systems, the MADT must include 1 PLIC entry.
- Entry ordering strongly can correlate with initialization order by an OS. The strong recommendation for the entry order of RINTC 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:
    - Composed 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 RINTC 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 - RINTC
    - Socket 0, Cluster 0, Thread 1 - RINTC
    - Socket 0, Cluster 1, Thread 0 - RINTC
    - Socket 0, Cluster 1, Thread 1 - RINTC
    - Socket 1, Cluster 0, Thread 0 - RINTC
    - Socket 1, Cluster 0, Thread 1 - RINTC
    - Socket 1, Cluster 1, Thread 0 - RINTC
    - Socket 1, Cluster 1, Thread 1 - RINTC

9.3.1.6. RHCT

- RISC-V Hart Capabilities Table
    - Communicates information about certain capabilities like ISA string.

9.3.1.8. 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. When doing so, the Interface Type values must follow the requirements in the table below.
- This specification requires revision 2 or later of the SPCR table. Revisions before 2 are not supported.
- The SPCR must be populated with correct ACPI GSIV interrupt routing information for the UART device.
- The SPCR console device must be included in the DSDT.

UART hardware	SPCR and DBG2 port type	DSDT _HID
16550-compliant	0x12	IMPLEMENTATION DEFINED

9.3.1.9. MCFG

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

9.3.1.10. 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.

9.3.2. Recommended ACPI Tables

ACPI tables that are recommended are listed in Appendix E.

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

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

9.3.3. Optional ACPI Tables

All other tables that are defined in the ACPI specification can be used as needed for RV64 platforms, but only if they comply with syntax and semantics of the specification.

9.4. ACPI Definition Blocks

Within the DSDT or SSDT tables that are used to describe the platform, devices are defined by ACPI definition blocks [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.

9.5. ACPI Methods and Objects

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

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

ACPI methods that are recommended are listed in Appendix F. Not every platform that is compliant with this specification provides all of these methods. This is because many methods reference optional platform features.

9.5.1. Global Methods and Objects

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

Platforms must not define processors using the global _PR (Processors) namespace. [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. [ACPI § 9.2.1]

9.5.2. Device Methods and Objects

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

- _ADR: Address on the parent bus of the device. Either this object or the _HID must be provided. This object is essential for PCI, for example. [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. [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. [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. [ACPI § 6.1.5]
- _STA: Status. This method identifies whether the device is on, off, or removed. [ACPI § 6.3.7][ACPI § 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. [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.

9.5.3. 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. [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. [ACPI § 5.6.5.3][ACPI § 5.6.4.1]

9.5.4. 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. [ACPI § 5.6.9.2]
- _EVT: Event method for interrupt-signaled interrupts. [ACPI § 5.6.9.3]

9.5.5. 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.

9.6. Hardware Requirements Imposed 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). [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]
    - **XXX** NMI-equivalent signal for use in fatal errors.
    - [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.

9.6.1. Process 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. [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].

XXX mechanism for monitoring performance of a logical processor (**XXX tying into a RISC-V FFH specification?**)

9.6.2. Time and Alarm Device

UEFI Runtime Service should be used to provide time services, unless the RTC is attached to an OS-managed bus (e.g. I2C with other OS-visible devices). See UEFI Real-time Clock for more. In situations where the TAD device depends on an OS driver for correct function (SPI, I2C, etc.), it is recommended the TAD device is implemented in a way to function if the OS driver is not loaded (e.g., when a driver is loaded for a dependency, ACPI invokes a method to bypass direct access and go through OS-backed OperationRegion).

10. SMBIOS Requirements

10.1. SMBIOS Version

This document references the following specification and versions: SMBIOS **XXX** Published (as a WIP) **XXX 202x**.

Legacy SMBIOS tables and formats are not supported.

10.1.1. 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 regio
  reclaimed by a UEFI-compliant operating system after UEFI ExitBootServices() is called

10.2. SMBIOS Structures

SMBIOS implementations vary by system design and form factor. For an OS-A-SEE-compliant system, the following SMBIOS structures are required or recommended. For required data within these structures, please refer to Table 4 and Annex A of the SMBIOS specification.
Unless specified otherwise in the SMBIOS specification, all “human readable” strings in SMBIOS structures must be UTF-8 encoded, using US-ASCII characters.
For additional guidance and requirements on SMBIOS tables reporting for telemetry and servicing, refer to [20].

10.2.1. Type00: BIOS Information (Required)

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

10.2.2. Type01: System Information (Required)

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

10.2.3. Type02: Baseboard (or Module) Information (Recommended)

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

10.2.4. Type03: System Enclosure or Chassis (Recommended)

- 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

10.2.5. 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
- 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 any of:
    - a discrete SoC
    - physical chip package (implementing a chip-to-chip extension of cache coherency)
    - chiplet implementing a set of processors / NUMA domain separate from other chiplets on a package

10.2.6. Type07: Cache Information (Required)

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

10.2.7. Type08: Port Connector Information (Recommended for Platforms with Physical Ports)

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

10.2.8. Type09: System Slots (Required for Platforms with Expansion Slots)

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

10.2.9. Type11: OEM Strings (recommended)

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

10.2.10. Type13: BIOS Language Information (Recommended)

- Installable Languages
- Flags
- Current Language

10.2.11. Type14: Group Associations (Recommended for Platforms To Describe Associations Between SMBIOS Types)

- Group Name
- Item Type
- Item Handle

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

10.2.12. Type16: Physical Memory Array (Required)

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

10.2.13. 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

10.2.14. Type19: Memory Array Mapped Address (Required)

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

10.2.15. Type32: System Boot Information (Required)

- Boot Status

10.2.16. Type38: IPMI Device Information (Required for Platforms with IPMIv1.0 BMC Host Interface)

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

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

10.2.17. Type39: System Power Supplies (Recommended for Servers)

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

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

10.2.18. Type41: Onboard Devices Extended Information (Recommended)

- Reference Designation
- Device Type

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

10.2.19. Type42: Redfish Host Interface (Required for Platforms Supporting Redfish Host Interface [16])

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

10.2.20. Type42: TPM Device (Required for Platforms with a TPM)

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

10.2.21. Type44: Standard Processor Additional Information (Required)

XXX this should list all fields once Type44 definitions are nailed down, right now it just lists what is absolutely minimal to have. See https://github.com/riscv/riscv-smbios/blob/main/riscv-smbios.adoc#standard-processor-additional-information-type-44-structure.

- ...
- Machine Vendor ID
- Machine Architecture ID
- Machine Implementation ID
- ...

10.2.22. Type45: Firmware Inventory Information (Recommended)

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

- Associated Components Handles

This SMBIOS type can be used to describe firmware components in the system, such as SBI (when separate from a UEFI implementation), SCP, NIC firmware, and so on. Systems that provide firmware inventory with a BMC (using Redfish for instance) and publish this SMBIOS type must correlate the data in this SMBIOS type with the data provided by the BMC.
Note: Firmware inventory provided in this SMBIOS type is a static snapshot taken at the end of the system boot. It does not reflect dynamic changes to firmware components versions due to out-of-band updates or hot plug of devices. Systems that support dynamic FW updates should not solely rely on this SMBIOS table for reporting FW inventory.
The entries in this table are platform specific, and not standardized. The following list provides a guidance on sample use cases for an SMBIOS Type 45 implementation:

- System Controllers
    - Could be multiple entries, describing different System Controllers in the system, depending on
      whether their FW is reported independently or not. Examples are SCP, SatMC or MCP, CXL
      Fabric Manager, and so on.
- M-Mode Firmware
    - Could be a single entry that covers everything (for UEFI implementations that don't rely on bootstrap from separate M-Mode SBI implementation), or multiple entries (SBI implementation, TEE, etc).
    - Would not exist in virtual machine environments.
- S-Mode Firmware
    - For UEFI implementations in virtual machine environments or that rely on bootstrap from separate M-Mode SBI implementation.
- Option Devices
    - Could be an entry per PCIe option device (such as a NIC) that presents a firmware version via UEFI Firmware Management Protocol (FMP)

10.2.23. Type46: String Property (Recommended)

- String Property ID
- String Property Value
- Parent handle

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

Appendices

Appendix A: Required UEFI Boot Services

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

EFI_SET_WATCHDOG_TIMER	7.5
EFI_CONNECT_CONTROLLER	7.3
EFI_DISCONNECT_CONTROLLER	7.3
EFI_OPEN_PROTOCOL	7.3
EFI_CLOSE_PROTOCOL	7.3
EFI_OPEN_PROTOCOL_INFORMATION	7.3
EFI_PROTOCOLS_PER_HANDLE	7.3
EFI_LOCATE_HANDLE_BUFFER	7.3
EFI_LOCATE_PROTOCOL	7.3
EFI_INSTALL_MULTIPLE_PROTOCOL_INTERFACES	7.3
EFI_UNINSTALL_MULTIPLE_PROTOCOL_INTERFACES	7.3
EFI_CALCULATE_CRC32	7.5
EFI_COPY_MEM	7.5
EFI_SET_MEM	7.5
EFI_CREATE_EVENT_EX	7.1

Appendix B: Required UEFI Runtime Services

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

EFI_UPDATE_CAPSULE	8.5
EFI_QUERY_CAPSULE_CAPABILITIES	8.5
EFI_QUERY_VARIABLE_INFO	8.2

Note: EFI_GET_WAKEUP_TIME and EFI_SET_WAKEUP_TIME must be implemented, but might simply return EFI_UNSUPPORTED. EFI_UPDATE_CAPSULE and EFI_QUERY_CAPSULE_CAPABILITIES must be implemented, but might simply return EFI_UNSUPPORTED. Services that are not implemented must be appropriately declared in the EFI_RT_PROPERTIES_TABLE.

Appendix C: Required UEFI Configuration Table Entries

Configuration Table
EFI_CONFORMANCE_PROFILE_TABLE_GUID

Appendix D: Optional and Conditionally Required UEFI Configuration Table Entries

Configuration Table
EFI_ACPI_20_TABLE_GUID
SMBIOS3_TABLE_GUID
EFI_DTB_TABLE_GUID
EFI_RT_PROPERTIES_TABLE_GUID
EFI_SYSTEM_RESOURCE_TABLE GUID (ESRT) (UEFI Configuration Table)

Appendix E: Required UEFI Protocols

E.1 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

E.2 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
EFI_BLOCK_IO_PROTOCOL	13.9
EFI_BLOCK_IO2_PROTOCOL	13.10

The Load File protocol is used to obtain files from arbitrary devices that are primarily boot options. This includes, for example, booting from network devices, The Load File 2 protocol is used to obtain files from arbitrary devices that are not boot options. The Block I/O and Block I/O 2 protocols are used for block device access by operating system bootloaders (e.g. with their own file system manipulation).

E.3 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

E.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

E.5 Random Number Generator Protocols

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.

E.6 RISC-V Protocols

Service	§
RISCV_EFI_BOOT_PROTOCOL	[3]

Appendix F: Optional and Conditionally Required UEFI Protocols

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

F.1 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

F.2 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

F.3 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.

F.4 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
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

F.5 VLAN Protocols

Service	UEFI §
EFI_VLAN_CONFIG_PROTOCOL	27.1

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

F.6 iSCSI Protocols

Service	UEFI §
EFI_ISCSI_INITIATOR_NAME_PROTOCOL	16.2

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

F.7 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

F.8 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.

F.9 Firmware Update

Service	UEFI §
EFI_FIRMWARE_MANAGEMENT_PROTOCOL	23.1

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 supports capsule-based firmware updates.

F.10 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

Required on platforms with CXL devices.

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

- Platforms supporting CXL 1.1 devices must follow the rules outlined in [CXL § 9.11], including the System
  Firmware view [CXL § 9.11.2], the System Firmware enumeration [CXL § 9.11.4], and the CXL device discovery flow [CXL § 9.11.5]
- Platforms supporting CXL 2.0 devices must follow the rules outlined in CXL 2.0 Enumeration [CXL § 9.12]
  and CXL OS Firmware Interface Extensions [CXL § 9.14]

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

F.11 PCIe UEFI Protocols

Service	UEFI §
EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL	14.2
EFI_PCI_IO_PROTOCOL	14.4

Required on platforms with PCIe devices.

F.12 Graphics Protocols

Service	UEFI §
EFI_GRAPHICS_OUTPUT_PROTOCOL	12.9.2

Required on platforms with framebuffers.

G Recommended and Conditionally Required ACPI Tables

G.1 I/O Topology

XXX depends on IOMMU and the ACPI table to support it.

G.2 TPM

ACPI table for describing a TPM2 device.

XXX need Start Method value for SBI ecall to support fTPM.

ACPI Signature	Full Name	TCG §
TPM2	TPM2 Table	8.3

G.3 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
RAS2	ACPI RAS2 Feature Table	ACPI code-first ECR

G.4 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

G.5 PCC

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

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

G.6 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

G.7 NVDIMM Firmware Interface

NFIT describes NVDIMM that is required if NVDIMM is supported or if UEFI-created RAM-disks are present.

ACPI Signature	Full Name	ACPI §

NFIT	NVDIMM Firmware Interface Table	5.2.25

G.8 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	

G.9 IPMI

SPMI describes the processor-relative, translated, fixed resources of an IPMI system interface at system boot time, if and only if IPMI has been implemented.

ACPI Signature	Full Name	ACPI §
SPMI	Server Platform Management Interface Table	https://uefi.org/acpi

G.10 CXL

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

ACPI Signature	Full Name	§
CEDT	CXL Early Discovery Table	CXL § 9.14.1

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

G.11 iSCSI and NVMe-oF

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

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

G.12 DBG2

Debug Port Table 2 (DBG2). See http://uefi.org/acpi

- This table provides a standard debug port.

- This table can be used to describe the UART. When doing so, the serial sub-type values must follow the requirements in 9.3.1.8. SPCR.
- If OS debugging through serial port is desired, the system must make the UART instance that is specified in DBG2 to be exclusively available for use by the debugger. How the system enables this is IMPLEMENTATION DEFINED.

Appendix H: Recommended and Conditionally Required ACPI Methods

This section lists recommended and conditionally required ACPI methods.

H.1 CPU Performance Control

For CPU performance and control, two mutually exclusive methods exist. Due to the CPPC requirement, only the newer _CPC method is supported in conjunction with the PCCT. As per ACPI spec, there is no support for mixed mode (CPPC & legacy PSS, _PCT, _PPC) operation.

Method	Full Name	ACPI §
_CPC	Continuous Performance Control (replaces _PCT and _PSS)	8.4.7.1

H.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

H.3 NUMA

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

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

H.4 IPMI

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	IPMI § 5.5.2.4.4
_SR	IPMIv2: the IPMI revision that is supported by the platform, if IPMI has been implemented	IPMI

H.5 Device Configuration and Control

The following methods provide configuration and control for devices.

Method	Full Name	ACPI §
_CLS	Class code (for non-PCI devices compatible with PCI drivers)	6.1.3
_CID	Compatible ID	6.1.2
_DSD	Device Specific Data: provides more device properties and information. Implementations are recommended to follow the guidance of the UEFI forum as outlined in the DSD implementation Guide [17]	6.2.5
_DSM	Device Specific Method (used to convey info to ACPI that it might not currently have a mechanism to describe)	9.1.1
_INI	Initialize a device	6.5.1

H.6 Resources

The following methods provide descriptions for devices.

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

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

H.7 CXL

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

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

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

H.8 Time and Alarm

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

H.9 PCI and PCIe

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

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

- If Steering Tags for cache locality are supported, then Function Index = 0Fh for Cache Locality TPH Features must be implemented, see TPH ECN[23].
- If the firmware reserves memory regions using Reserved Memory Range nodes in the **XXX ACPI IOMMU 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.

Appendix I: CXL Requirements

I.1 CXL Host Bridge

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

I.2 CXL Root Device

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

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

I.2.1 ACPI Device Object for CXL Root Device

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

Note: The CXL Root Device HID value is defined in an ACPI code-first ECR (**XXX it's probably in the spec already**)

- For each CXL root device whose properties need to be conveyed to the OS, the firmware must include the following methods as a child of the root device:
  _DSM, function for retrieving QoS Throttling Group (QTG) IDs

I.3 NUMA

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

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

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

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

Note: The SRAT Generic Port entry is defined in an ACPI code-first ECR (**XXX probably in the spec already**).

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.