Arm® Server Base Boot Requirements 1.2

Platform Design Document

Document number: DEN0044D

Server 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
16 August 2014	A	Non-Confidential	Initial release, SBBR version 0.9
8 March 2016	B	Non-Confidential	SBBR version 1.0 Updated referenced specifications to: UEFI 2.5, ACPI 6.0, SMBIOS 3.0.0
31 May 2018	C	Non-Confidential	SBBR version 1.1 • Updated referenced specifications to: UEFI 2.7, ACPI 6.2, SMBIOS 3.1.1 • ACPI Interrupt-signaled Events support • ACPI GED support • Secondary core boot standardization • PSCI minimum revision back to 1.0 • SMBIOS Processor Information • SMBIOS structure data requirements clarification • Secure and Trusted Boot • Secure Firmware Update • UEFI REST Protocol support • UEFI Capsule Service clarification • ACPI PCI IO Address Translation clarifications • UEFI PCI Root Bridge IO Protocol Address Translation clarifications • UEFI GOP implementation clarifications • IORT implementation guidelines • SPMI recommendation removal • SMBIOS Redfish Host Interface support • Native AArch64 image requirements for UEFI applications and drivers • Clarifications of SSDT being optional • Clarifications on UEFI Load File and Load File 2 Protocols • UEFI Random Number Generator Protocol guideline • ACPI MCFG Table guideline clarifications
September 2, 2019	D	Non-Confidential	SBBR version 1.2 • ACPI Specification version update • UEFI Specification version update • PPTT move from recommended to mandatory • MADT recommended ordering • ESRT vlarification • PE/COFF image guidance • CPPC Requirement

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

1 ABOUT THIS DOCUMENT

1.1 Introduction

This Server Base Boot Requirements (SBBR) specification is intended for SBSA[2]-compliant 64-bit Armv8 servers. It defines the base firmware requirements for out-of-box support of any Arm SBSA-compatible Operating System or hypervisor. The requirements in this specification are intended to be minimal yet complete for booting a multi-core Armv8 server platform, while leaving plenty of room for OEM or ODM innovations and design details.

This specification is intended to be OS-neutral. It leverages the prevalent industry standard UEFI and ACPI firmware specifications.

1.2 References

This document refers to the following documents:

Reference	Doc No	Authors	Title
[1]	Arm DDI 0487	Arm	Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile
[2]	Arm DEN 0029	Arm	Server Based System Architecture SBSA Version 6.0
[3]	ACPI 6.3	UEFI.org	Advanced Configuration and Power Interface Specification. Revision 6.3
[4]	UEFI Specification 2.8	UEFI.org	Unified Extensible Firmware Interface Specification. Version 2.8
[5]	Arm DEN 022	Arm	Power State Coordination Interface (PSCI) Version 1.1
[6]	SMBIOS Version 3.2.0	DMTF	System Management BIOS (SMBIOS) Reference Specification
[7]	Arm DEN 0054	Arm	Software Delegated Exception Interface (SDEI)
[8]	Redfish Host Interface Specification Version 1.1.0	DMTF	Redfish Host Interface Specification Version 1.1.0
[9]	Arm DEN 0006	Arm	Arm TBBR Specification
[10]	TCG PC Firmware Profile v2.0	TCG	TCG PC Client Platform Firmware Profile Specification Family 2.0
[11]	TCG ACPI	TCG	TCG ACPI Specification for TPM Family 1.2 and 2.0
[12]	TCG EFI	TCG	TCG EFI Protocol Specification, Family 2.0
[13]	TCG PPI	TCG	TCG PC Client Platform Physical Presence Interface Specification Family 1.2 and 2.0
[14]	TCG PTP	TCG	TCG PC Client Platform TPM Profile (PTP) Specification Family 2.0
[15]	TCG RAM	TCG	TCG Platform Reset Attack Mitigation Specification Version 1.00
[16]	NIST800-147B	NIST	BIOS Protection Guidelines for Servers

[17]	NIST800-193	NIST	Platform Firmware Resiliency Guidelines
[18]	NIST800-155	NIST	BIOS Integrity Measurement Guidelines
[19]	PCI FW	PCI SIG	PCI Firmware Specification Revision 3.2

1.2.1 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 [3] section 5.6.6

UEFI § 6.1 - Reference to the UEFI specification [4] section 6.1

1.3 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.
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.
SiP	Silicon Partner. In this document, the silicon manufacturer.
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.

1.4 Feedback

Arm welcomes feedback on its documentation.

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

• The title Server Base Boot Requirements.

• The document ID and version DEN0044C 1.1.

• The page numbers to which your comments apply.

• A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

2 SCOPE

This document defines the Boot and Runtime Services expected by an enterprise platform Operating System or hypervisor, for an SBSA-compliant Arm AArch64 server which follows the UEFI and ACPI specifications.

The UEFI and ACPI specifications have been chosen to ease the adoption of Arm, by aligning the AArch64 server ecosystem to the existing enterprise server market. Many other AArch64 systems exist within other market segments, but their boot and firmware choices are beyond the scope of this document.

This document references the following specification and versions:

UEFI 2.8 Published March 2019, includes the AArch64 bindings.

ACPI 6.3 Published January 2019, includes the Reduced HW profile.

This specification defines the boot and Runtime Services for a physical system, including services that are required for virtualization. It does not define a standardized abstract virtual machine view for a Guest Operating System.

When present with in a system, this document makes additional references to the Power State Coordination Interface:

PSCI 1.1 Published April 2017.

3 UEFI

3.1 UEFI Version

Boot and system firmware for 64-bit Arm servers is based on the UEFI specification[4], version 2.8 or later, incorporating the AArch64 bindings.

3.2 UEFI Compliance

Any UEFI-compliant system must follow the requirements that are laid out in section 2.6 of the UEFI specification. However, to ensure a common boot architecture for server-class AArch64, systems compliant with this specification must always provide the UEFI services and protocols that are listed in Appendix A, Appendix B , and Appendix C of this document.

3.3 UEFI System Environment and Configuration

3.3.1 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 or not virtualization is used or supported.

Resident UEFI firmware might target a specific Exception level. In contrast, UEFI Loaded Images, such as third-party drivers and boot applications, must not contain any built-in assumptions of the exception level to be loaded at boot time, since they can be loaded into EL1 or EL2.

3.3.1.1 UEFI Boot at EL2

Most systems are expected to boot UEFI at EL2, to allow for the installation of a hypervisor or a virtualiazation-aware Operating System.

3.3.1.2 UEFI Boot at EL1

Booting of UEFI at EL1 is most likely within a hypervisor hosted 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.

3.3.2 System Volume Format

The system firmware must support GPT Partitioning.

3.3.3 UEFI Image Format

UEFI allows the extension of platform firmware by loading UEFI driver and UEFI application images [UEFI § 2]

3.3.3.1 UEFI Drivers

If a platform supports the inclusion or addition of any device that provides a container for one or more UEFI drivers [UEFI § 2.1.4] required for the initialization of that device, then at least one of the UEFI drivers must be in the A64 binary format to be used for the systems complying to this specification.

3.3.3.2 UEFI Applications

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

3.3.4.3 PE/COFF Image

The SectionAlignment and FileAlignment fields as defined in Microsoft PE Format (https://docs.microsoft.com/en-

us/windows/desktop/Debug/pe-format#section-data) must contain the value of at least 4KiB. Higher values are also permitted, for example for DXE_RUNTIME_DRIVER modules, to meet the 64 KiB 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 4 KiB should not contain any sections that have both the IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE attributes set.

3.3.4 GOP Protocol

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

3.3.5 Address Translation Support

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

3.4 UEFI Boot Services

3.4.1 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, which means that virtual addresses must have equal physical addresses.

The default RAM allocated attribute must be `EFI_MEMORY_WB`.

3.4.2 UEFI Loaded Images

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

3.4.3 Configuration Tables

A UEFI system that complies with this specification must provide the following tables via 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 4.
- `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 5.

3.5 UEFI Runtime Services

UEFI Runtime Services exist after ExitBootServices() is called and are designed to 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.

3.5.1 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.

3.5.2 Runtime Memory Map

Before calling ExitBootServices(), the final call to GetMemoryMap() returns a description of the entire UEFI memory map which 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 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, UEFI Specification constrains all mapped 4KiB memory pages to have identical page attributes within the same physical 64K page.

3.5.3 Real-time Clock

The Real-time Clock must be accessible via 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.

3.5.4 UEFI Reset and Shutdown

The UEFI Runtime service ResetSystem() must implement the following commands, 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.

Note: When Runtime Services and PSCI co-exist, it is anticipated that Operating System calls to reset the system will go via Runtime Services and not directly to PSCI.

3.5.5 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: This normally requires dedicated storage for UEFI variables that is not directly accessible from the Operating System.

3.6 Secure and Trusted Boot

3.6.1 Secure Boot

In this specification, Secure Boot refers to the mechanism for achieving a complete authentication and validation of all the executed software/firmware images.

If Secure Boot is implemented, the system must provide an implementation of a complete cascading Chain of Trust, from the initial firmware up to the first normal world firmware (Arm TBBR Specification[9] provides an example reference indication of such a Chain of Trust. A reference software implementation of the TBBR is available in the Arm Trusted Firmware code base located at https://github.com/ARM-software/arm-trusted-firmware). In order to establish an immutable root-of-trust, the initial firmware (i.e. the first instructions executed on SoC by the boot CPU or micro-controller) must be in an immutable read-only location. UEFI Secure Boot then continues the verification of the subsequent UEFI images until the launch of the OS loader. During this stage, the platform must conform to the Secure Boot definitions in the UEFI specification. After this, the OS can continue this chain of trust forward. All firmware images loaded from external boot media and executed on any processor or microcontroller on the SoC must be authenticated.

3.6.2 TCG Trusted Boot

TCG defines a Trusted Boot mechanism where integrity measurements of the software/firmware images are recorded into the Platform Configuration Registers (PCRs) on the TPM for the purpose of attestation. In addition, any modifiable configuration data stored on the external boot media must be measured.

Note: SBSA/SBBR Specifications support a TPM implementation that is compliant to TPM Library Specification, Family 2.0.

If TCG Trusted Boot is supported, the platform must provide support for TCG PC Client Platform Firmware Profile Specification Family 2.0[10], TCG ACPI Specification for TPM Family 1.2 and 2.0[11], TCG EFI Protocol Specification for TPM Family 2.0[12] and TCG PC Client Platform Physical Presence Interface Specification Family 1.2 and 2.0[13].

Note: TCG ACPI Specification for TPM defines a TPM Start Method based on vendor-specific Arm Secure Monitor Call(SMC). The OSes get the SMC function ID shown in Table 9, no additional parameters are involved in this SMC call. This supports Locality 0 for CRTM/SRTM on Arm systems.

Optionally, the platform can support the TCG PC Client Platform TPM Profile (PTP) Specification Family 2.0[14] for software interacting with the TPM.

Optionally, the platform may support the TCG Platform Reset Attack Mitigation Specification[15] for clearing memory upon unexpected resets and reboots.

Note: When the UEFI MemoryOverwriteRequestControl variable, ACPI _DSM method and PSCI memory protection API co-exist, it is anticipated that Operating System calls to mitigate the reset attack will go via the UEFI/ACPI interfaces and not directly to PSCI.

3.6.3 Relationships

Secure Boot and TCG Trusted Boot are independent of each other, and may be adopted individually. They are, however, complementary, and can opportunistically leverage each other when present in combination - e.g. UEFI Secure Boot secure variables can be stored on a TPM.

3.7 Secure Firmware Update

In this specification, Secure Firmware Update includes the update of pre-UEFI firmware (e.g., Arm Trusted Firmware or equivalent), system firmware (e.g., UEFI), and device firmware.

The TBBR Specification provides its Firmware Update definition in terms of Trusted and Non-trusted firmware responsibilities. Arm Trusted Firmware provides a reference software implementation as described at:

https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/firmware-update.rst

Secure Firmware Update refers to the mechanism for achieving a complete authentication and validation of the updated firmware implemented in a Root of Trust for Update (RTU).

If Secure Firmware Update is implemented, the requirements in the NIST 800-147B Specification[16] and the NIST 800-193 Specification[17] must be met.

Firmware update can be initiated from the host (e.g., OS or UEFI) or the Baseboard Management Controller (BMC). One or both of these mechanisms should be provided.

3.7.1 Host-initiated Firmware Update

Supporting host-initiated firmware update is optional.

If implemented, the following requirements must be met.

3.7.1.1 UEFI ESRT

System firmware must present a UEFI ESRT configuration table containing a system resource entry describing each updatable firmware component, including system firmware, device firmware, or UEFI drivers.

Note: It is permissible to implement a firmware resource entry that describes multiple firmware components, for example all system and device firmware, as a single monolithic entity. This allows a single update capsule to update multiple components.

3.7.1.2 UEFI Capsule Services

System firmware must implement the UpdateCasule() and QueryCapsuleCapabilities() runtime service that support the system firmware updates, as well as UEFI-updatable device firmware. UpdateCapsule() is used to pass the firmware update payload between the OS and the system firmware. It must recognize a firmware update payload passed to it before initiating the update process. However, peripheral devices may not be present during boot. These devices can have their firmware updated via OS runtime device drivers.

3.7.1.3 UEFI Firmware Management Protocol

UEFI drivers must provide the Firmware Management Protocol. This enables a standard mechanism for the device firmware update at the UEFI boot time. When mapped with ESRT and the UpdateCapsule() service, this provides a standard mechanism for runtime updating the UEFI-updatable device firmware.

3.7.2 BMC-initiated Firmware Update

Supporting BMC-initiated firmware update is optional and system implementation dependent.

If implemented, the requirements in the NIST 800-147B Specification[16] and NIST 800-193 Specification[17] must be met.

3.8 Platform Firmware Resiliency

To support resiliency of platforms against potentially destructive attacks, three principles are followed: Protection, Detection and Recovery. If implemented, the requirements in the NIST 800-193 Specification[17] must be met.

4 ACPI REQUIREMENTS

SBSA-compliant servers use the *Advanced Configuration and Power Interface* (ACPI) to describe the hardware resources that are installed, and to handle aspects of runtime system configuration, event notification, and power management.

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

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

4.1 ACPI Provided Data Structures

All platforms compliant with this specification must:

- Conform to the ACPI specification[3], 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).

4.2 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. The ACPI tables might be stored in ROM or flash memory as decided by the platform designers.

All platforms compliant 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 via 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.

4.2.1 Mandatory ACPI Tables

The following tables are mandatory for all compliant systems.

4.2.1.1 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.

4.2.1.2 XSDT

- *Extended System Description Table* (XSDT), ACPI § 5.2.8.

- The RSDP must contain a pointer to this table.
- This table, in turn, contains pointers to all other ACPI tables that are to be used by the OSPM.

4.2.1.3 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.
  - It is recommended that one of the server profiles (ACPI § 5.2.9.1) is selected.
  - The ARM_BOOT_ARCH flags describe the presence of PSCI. See ACPI § 5.2.9.4.

4.2.1.4 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 this table can be used to provide additional definition blocks if necessary.

4.2.1.5 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.
- It is strongly recommended that the entry order of GICC structures reflect 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 shared by all logical processors in the socket.
  - Each socket contains two processor clusters, and within each cluster there is cache 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

4.2.1.6 GTDT

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

4.2.1.7 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 Arm SBSA Generic UART.

4.2.1.8 SPCR

- *Serial Port Console Redirection* (SPCR). See http://uefi.org/acpi
    - This table provides the essential configuration information that is needed for headless operations, such as a kernel shell or console.
    - This table defines a Serial Port type, location, and interrupts.
    - Note: this table can be used to describe the Arm SBSA Generic UART.
- This specification requires revision 2 or above 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.

4.2.1.9 MCFG

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

4.2.1.10 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.

4.2.2 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, because many tables reference optional platform features.

Examples:

- 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.

4.2.3 Optional ACPI Tables

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

4.3 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 additional information. For example, processors must be described by definition blocks, whereas PCI devices are enumerated by a defined protocol.

4.4 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, that is, they are defined 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.

4.4.1 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 compliant 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 such as an LED.
    - See ACPI § 9.2.1

4.4.2 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. It 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, such as devices that are DMA capable.
    - See ACPI § 6.2.17
- _CRS: Current Resource Settings. This method provides essential information to describe resources, such as registers and their locations that are provided by the device.
    - See ACPI § 6.2.2
- _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.

4.4.3 GPIO Controllers

The HW-Reduced ACPI model has specific requirements for GPIO controllers and devices. Platforms supporting GPIO-signaled events 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 less than 255, the _Exx or _Lxx methods can be used instead.
  - See ACPI § 5.6.5.3 and 5.6.4.1.

4.4.4 Generic Event Devices

The HW-Reduced ACPI model has specific requirements for Generic Event Devices. Platforms supporting interrupt-signaled 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.

4.4.5 Address Translation Support

This specification recommends that PCIe-compliant devices are used, eliminating the need to support legacy IO port space. However, if the platform supports legacy IO port space, it must report the host (CPU) to PCI I/O bus address space translations using resource descriptors of type *DWordIO*, *QWordIO* or *ExtendedIO.* *TranslationType* must be set to *TypeStatic* (due to existing OS behaviour).

4.5 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.

What is reduced, is the complexity of the OSPM in supporting ACPI. For example, many requirements from versions earlier than version 5.0 can be ignored. At the same time, 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 compliant with this specification must provide the following *GPIO-Signaled* 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)[7] or one NMI-equivalent signal for use in fatal errors.
  - See ACPI § 18.

- At least one wake signal, which is routed via a platform event. Note: for systems that do not support Sx states except S5 soft off, this can be just the power button.

4.5.1 Processor Performance Control

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

This specification highly recommends that the Platform Communications Channel (PCC) be used for processor performance management. See ACPI § 14.

Note: An interrupt must be defined specifically for the doorbell.

4.5.2 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.

5 SMBIOS

The System Management BIOS (SMBIOS) 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.

5.1 SMBIOS Base Requirements

The SMBIOS table is required to conform to the SMBIOS specification[6], version 3.2.0 or later. Legacy SMBIOS tables and formats are not supported.

5.1.1 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.

5.2 SMBIOS Structures

SMBIOS implementations vary by server design and form-factor. For an SBBR-compliant server, 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.

5.2.1 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

5.2.2 Type01: System Information (REQUIRED)

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

5.2.3 Type02: Baseboard (or Module) Information (RECOMMENDED)

- Manufacturer
- Product
- Version

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

5.2.4 Type03: System Enclosure or Chassis (REQUIRED)

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

5.2.5 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 Type4 structure must be provided for every socket in the system, for example, N Type4 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[1].

5.2.6 Type07: Cache Information (REQUIRED)

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

5.2.7 Type08: Port Connector Information (RECOMMENDED for platforms with physical ports)

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

5.2.8 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

5.2.9 Type11: OEM Strings (RECOMMENDED)

- Count

5.2.10 Type13: BIOS Language Information (RECOMMENDED)

- Installable Languages
- Flags
- Current Language

5.2.11 Type15: System Event Log (RECOMMENDED)

5.2.12 Type16: Physical Memory Array (REQUIRED)

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

5.2.13 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

5.2.14 Type19: Memory Array Mapped Address (REQUIRED)

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

5.2.15 Type32: System Boot Information (REQUIRED)

- Boot Status

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

- IPMI Specification Revision
- I2C Slave 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

5.2.17 Type41: Onboard Devices Extended Information (RECOMMENDED)

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

5.2.18 Type42: Redfish Host Interface (REQUIRED for platforms supporting Redfish Host Interface[8])

- Interface Type
- Interface Specific Data
- Protocol Records

6 SECONDARY CORE BOOT

UEFI is defined as a uniprocessor specification that only uses a single CPU core for booting.

Platforms providing EL3 must implement the Power State Coordination Interface (PSCI) [5]. 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 this:

- 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.

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.

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_BIS_PROTOCOL	24.5
EFI_MTFTP4_PROTOCOL	30.3
EFI_MTFTP6_PROTOCOL	30.4

EFI_BIS_PROTOCOL is optional on machines that do not support Secure Boot.

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, such as a HDD. This configuration is intended for thin clients and *compute-only* nodes.

REST Protocol

Service	UEFI §
EFI_REST_PROTOCOL	29.7

Support for REST protocol is required on machines that support RESTful communication over HTTP (e.g., Redfish Host Interface to a BMC).

APPENDIX E RECOMMENDED ACPI TABLES

I/O Topology

ACPI Signature	Full Name	ACPI §
IORT	Support for SMMU, ITS, and system topology description	http://uefi.org/acpi

Table to describe 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.

Platform Error Interfaces

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	http://uefi.org/acpi

Tables that are required to support ACPI Platform Error Interfaces (APEI), which convey error information to the Operating System.

NUMA

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

Tables to describe topology and resources that are required by NUMA systems.

Platform Communications Channel (PCC)

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

Provides the interface to communicate to an on-platform controller.

Platform Debug Trigger

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

NVDIMM Firmware Interface

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

Table to describe NVDIMM that is required if NVDIMM is supported.

APPENDIX F RECOMMENDED ACPI METHODS

CPU performance control

For CPU performance and control, there are two mutually exclusive methods defined.

Method	Full Name	ACPI §
_CPC	Continuous Performance Control (replaces _PCT and _PSS)	8.4.6.1

Newer _CPC method in conjunction with the PCCT

_PSS	Performance Supported States (Superseded by _CPC)	8.4.6.2

Older _PSS method.

CPU and system idle control

For CPU and system idle management, the following method has been introduced in ACPI6.0.

Method	Full Name	ACPI §
_LPI	Low Power Idle States	8.4.4

NUMA

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

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

Device Configuration and Control

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 additional device properties and information.	6.2.5

	All compliant _DSD UUIDs and associated definitions must be published by the UEFI Forum, see http://www.uefi.org/acpi	
_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

Method	Full Name	ACPI §
_MLS	Human readable description in multiple languages. **Note**: this is preferred over _STR.	6.1.7
_PRS	Possible Resource Settings	6.2.12
_SRS	Set Resource Settings	6.2.16
_STR	Device description (in a single language). Superseded by _MLS.	6.1.10