RISC-V Boot and Runtime Services Specification (BRS)

BRS Task Group

Version 1.0, 29th August 2025: This document is Ratified.

Table of Contents

Preamble

This document is in the Ratified state

No changes are allowed. Any necessary or desired modifications must be addressed through a follow-on extension. Ratified extensions are never revised.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by (in alphabetical order):

Aaron Durbin, Andrei Warkentin, Andrew Jones, Anup Patel, Atish Patra, Beeman Strong, Darius Rad, Heinrich Schuchardt, Haibo Xu, Jamie Iles, John Hauser, Radim Krčmář, Rahul Pathak, Paul Walmsley, Samuel Holland, Sia Jee Heng, Sunil V L, Vedvyas Shanbhogue

Chapter 1. Introduction

The *RISC-V Boot and Runtime Services Specification* (BRS) defines a standardized set of software capabilities, that portable system software, such as operating systems and hypervisors, can rely on being present in an implementation to utilize in acts of device discovery, OS boot and hand-off, system management, and other operations.

The BRS specification is targeting systems that implement S/U privilege modes, and optionally the HS privilege mode. This is the expected deployment for OSVs and system vendors in a typical ecosystem covering client systems up through server systems where software is provided by different vendors than the system vendor.

This specification standardizes the requirements for software interfaces and capabilities by building on top of relevant industry and ratified RISC-V standards.

1.1. Releases

It is expected that the BRS will periodically release a new specification. The determination of a new release will be based on the evaluation of significant changes to its underlying dependencies.

1.2. Approach to Solutions

The BRS focuses on two solutions in the form of what is deemed a recipe. Each recipe contains the requirements needed to fulfill each solution. The requirements of each recipe will be marked accordingly with a unique identifier. The recipes are BRS-I (Interoperable) and BRS-B (Bespoke).

1.3. Testing and Conformance

To be compliant with this specification, an implementation MUST support all mandatory rules and MUST support the listed versions of the specifications. This standard set of capabilities MAY be extended by a specific implementation with additional standard or custom capabilities, including compatible later versions of listed standard specifications. Portable system software MUST support the specified mandatory capabilities to be compliant with this specification.

The rules in this specification use the following format:

ID#	Rule
`CAT_NNN`	The `CAT` is a category prefix that logically groups the rules and is followed by 3 digits - `NNN` - assigning a numeric ID to the rule. The rules use the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" that are to be interpreted as described in RFC 2119 [1] when, and only when, they appear in all capitals, as shown here. When these words are not capitalized, they have their normal English meanings.

ID#	Rule
A rule or a group of rules may be followed by non-normative text providing context or justification for the rule. The non-normative text may also be used to reference sources that are the origin of the rule.	

1.4. Glossary

Most terminology has the standard RISC-V meaning. This table captures other terms used in the document. Terms in the document prefixed by **PCIe** have the meaning defined in the *PCI Express Base Specification* [2] (even if they are not in this table).

Table 1. Terms and definitions

Term	Definition
ACPI	*Advanced Configuration and Power Interface Specification* [3].
BRS	*RISC-V Boot and Runtime Services Specification*. This document.
BRS-I	Boot and Runtime Services recipe targeting interoperation across different software suppliers.
BRS-B	Boot and Runtime Services recipe using a bespoke solution.
DT	*Device Tree* [4].
EBBR	*Embedded Base Boot Requirements Specification* [5].
OSV	Operating System Vendor.
OS	Operating System or Hypervisor.
Profile	*RISC-V Profile* [6].
RPMI	*RISC-V Platform Management Interface* [7].
RVI	RISC-V International.
SBI	*RISC-V Supervisor Binary Interface Specification* [8].
SMBIOS	*System Management BIOS (SMBIOS) Reference Specification* [9].
SoC	System on a chip, a combination of processor and supporting chipset logic in single package.
UEFI	*Unified Extensible Firmware Interface Specification* [10].

Chapter 2. Recipes

In this context, a recipe is a collection of firmware specification requirements that hardware, firmware, and software providers can implement to increase the likelihood that software written to the recipe will run predictably on all conforming devices.

The BRS specification defines two recipes: BRS-I (for "Interoperable") and BRS-B (for "Bespoke").

2.1. BRS-I Recipe

The BRS-I recipe aims to simplify end-user experiences, software compatibility and OS distribution, by defining a common specification for boot and runtime interfaces. BRS-I is expected to be used by general-purpose compute devices such as servers, desktops, laptops and other devices with industry expectations on silicon vendor, OS and software ecosystem interoperability. BRS-I enables operating system providers to build a single **generic** operating system image that can be **successfully booted** on compliant systems. **Generic** means not requiring system-specific customizations - only an implementation of BRS-I requirements. **Successfully boot** means basic system configuration, sufficient for detecting the need for system-specific drivers and loading such drivers.

It is understood that systems will deliver features beyond those covered by BRS-I. However, software written against a specific version of BRS-I must run, unaltered, without **anomalous and unexpected behavior** on systems that include such features and that are compliant to that specific version of BRS-I. Such behavior, caused by factors entirely unknown to a generic OS, is hard to diagnose and always results in a terrible user experience that negatively affects the value of the whole RISC-V standards-based ecosystem. **Anomalous and unexpected behavior** is taken to mean system instability and worst-case behavior for non-specialized workloads, but does not include suboptimal/unoptimized behavior or missing I/O or accelerator drivers. Any additional firmware features that cause anomalous and unexpected behavior must be disabled by default, and only enabled by opt-in. See additional guidance.

Table 2. BRS-I Recipe Overview

Profile	UEFI	ACPI	DT	SBI	SMBIOS
>= RVA20S64	>= 2.10	>= 6.6	optional, >= v0.3	>= 2.0	>= 3.7.0

2.2. BRS-B Recipe

BRS-B is intended for cases where only a minimal level of firmware interaction is mandated, focusing primarily on the boot process. The BRS-B recipe is the simpler of the two BRS recipes. It is expected to be used by software that is tailored to specific devices. Examples include many types of mobile devices, devices with real time response requirements, or embedded devices running rich operating systems with custom distributions.

Table 3. BRS-B Recipe Overview

Profile	UEFI	ACPI	DT	SBI	SMBIOS
>= RVA20S64	EBBR, >= 2.1.0 [5]	optional, >= 6.6	optional, >= v0.3	>= 2.0	optional, >= 3.7.0

Either ACPI or DT may be used to describe hardware to the OS, but never both at the same time.

Chapter 3. Hart Requirements

A compliant system includes a RISC-V application processor and the requirements in this section apply solely to harts in the application processors of a system.

The BRS specification is minimally prescriptive on the RISC-V hart requirements. It is anticipated that detailed requirements will be driven by target market segment and product/solution requirements.

ID#	Rule
`HR_010`	The RISC-V application processor harts MUST be compliant to `RVA20S64` profile [6].
The BRS governs the interactions between 64-bit OS supervisor-mode software and 64-bit firmware. These are minimum requirements allowing for the wide variety of existing and future hart implementations to be supported. It is expected that operating systems and hypervisors may impose additional profile/ISA requirements, depending on the use-case and application.	

Chapter 4. SBI Requirements

The *RISC-V Supervisor Binary Interface Specification* (SBI) [8] defines an interface between the supervisor mode and the next higher privilege mode. This section defines the mandatory SBI version and extensions implemented by the higher privilege mode in order to be compatible with this specification.

ID#	Rule
`SBI_010`	The SBI implementation MUST conform to SBI v2.0 or later.
`SBI_020`	The SBI implementation MUST implement the Hart State Management (HSM) extension.
HSM is used by an OS for starting up, stopping, suspending and querying the status of secondary harts.	

Certain requirements are conditional on the presence of RISC-V ISA extensions or system features.

ID#	Rule
`SBI_030`	The Timer Extension (TIME) MUST be implemented, if the RISC-V "stimecmp / vstimecmp" Extension (Sstc, [11]) is not available.
`SBI_040`	The S-Mode IPI Extension (sPI) MUST be implemented, if the Incoming MSI Controller (IMSIC, [12]) is not available.
`SBI_050`	The RFENCE Extension (RFNC) extension MUST be implemented, if the Incoming MSI Controller (IMSIC, [12]) is not available.
`SBI_060`	The Performance Monitoring Extension (PMU) MUST be implemented, if the counter delegation-related S-Mode ISA extensions (Sscsrind [13] and Ssccfg [14]) are not present.
`SBI_070`	The Debug Console Extension (DBCN) MUST be implemented if the ACPI SPCR table references Interface Type 0x15.

Chapter 5. BRS-I UEFI Requirements

The *Unified Extensible Firmware Interface Specification* (UEFI) describes the interface between the OS and the supervisor-mode firmware.

This section defines the BRS-I mandatory and optional UEFI rules on top of existing [10] specification requirements. Additional non-normative guidance may be found in the firmware implementation guidance section.

All content in this section is optional and recommended for BRS-B.

ID#	Rule
`UEFI_010`	MUST implement a 64-bit UEFI firmware.
`UEFI_020`	MUST meet the 3rd Party UEFI Certificate Authority (CA) requirements on UEFI memory mitigations [15].
`UEFI_030`	MUST meet the following memory map rules: • The default memory space attribute is `EFI_MEMORY_WB`. • Paged virtual-memory scheme MUST be configured by the firmware with identity mapping and MUST support `EFI_MEMORY_ATTRIBUTE_PROTOCOL` protocol. • Only use `EfiRuntimeServicesData` memory type for describing any SMBIOS data structures.
Paged virtual memory scheme is required for platform protection use cases before handing off to the OS.	
`UEFI_040`	An implementation MAY comply with the *UEFI Platform Initialization Specification* [16].
`UEFI_050`	All hart manipulation internal to a firmware implementation SHOULD be done before completion of the `EFI_EVENT_GROUP_READY_TO_BOOT` event. Firmware MUST place all secondary harts in an offline state before completion of the EFI_EVENT_GROUP_READY_TO_BOOT event.
This ensures an OS loader is entered with an OS-compatible state for all harts.The OS loader and/or the OS may resume the secondary harts, if required, as part of their boot and join sequence.	
`UEFI_060`	The implementation MUST declare the `EFI_CONFORMANCE_PROFILES_UEFI_SPEC_GUID` conformance profile.
The `EFI_CONFORMANCE_PROFILES_UEFI_SPEC_GUID` *conformance profile MUST be declared, as the BRS requirements are a superset of UEFI [10] (Section 2.6).*	
`UEFI_070`	The implementation MUST declare the `EFI_CONFORMANCE_PROFILE_BRS_1_0_SPEC_GUID` conformance profile `({ 0x05453310, 0x0545, 0x0545, { 0x05, 0x45, 0x33, 0x05, 0x45, 0x33, 0x05, 0x45 }})`.
Only a system fully compliant to the requirements in this section MUST declare the `EFI_CONFORMANCE_PROFILE_BRS_1_0_SPEC_GUID` *conformance profile.*	

<table>
<tr><th>ID#</th><th>Rule</th></tr>
<tr><td><code>UEFI_080</code></td><td>A Device Tree MUST only be exposed to the OS if no actual hardware description is included in the DT.</td></tr>
<tr><td colspan="2">Such a "dummy" DT could be installed by firmware, as a UEFI configuration table entry of type <code>EFI_DTB_TABLE_GUID</code>, to provide necessary hand-off info to an OS, for example, to provide RAM disk information (e.g. via <code>/chosen/linux,initrd-start</code>).</td></tr>
</table>

5.1. BRS-I I/O-specific Requirements

<table>
<tr><th>ID#</th><th>Rule</th></tr>
<tr><td><code>UIO_010</code></td><td>Systems implementing PCIe MUST always initialize all root complex hardware and perform resource assignment for all endpoints and usable hotplug-capable switches in the system, even in a boot scenario from a non-PCIe boot device.</td></tr>
<tr><td colspan="2">This is a stronger requirement than the PCI Firmware Specification firmware/OS device hand-off state ([17] Section 3.5). See additional guidance.</td></tr>
<tr><td><code>UIO_020</code></td><td>Systems implementing <code>EFI_GRAPHICS_OUTPUT_PROTOCOL</code> SHOULD configure the frame buffer to be directly accessible.</td></tr>
<tr><td colspan="2">That is, <code>EFI_GRAPHICS_PIXEL_FORMAT</code> is not <code>PixelBltOnly</code> and <code>FrameBufferBase</code> is reported as a valid hart memory-mapped I/O address.</td></tr>
</table>

5.2. BRS-I UEFI Runtime Services

<table>
<tr><th>ID#</th><th>Rule</th></tr>
<tr><td><code>URT_010</code></td><td>Systems without a Real-Time Clock (RTC), but with an equivalent alternate source for the current time, MUST meet the following requirements:
<code>GetTime()</code> MUST be implemented.
<code>SetTime()</code> MUST return <code>EFI_UNSUPPORTED</code>, and be appropriately described in the <code>EFI_RT_PROPERTIES_TABLE</code>.</td></tr>
<tr><td colspan="2">See additional guidance.</td></tr>
<tr><td><code>URT_020</code></td><td>Systems with a Real-Time Clock on an OS-managed bus (e.g. I2C, subject to arbitration issues due to access to the bus by the OS) MUST meet the following requirements:
<code>GetTime()</code> and <code>SetTime()</code> MUST return <code>EFI_UNSUPPORTED</code>, when called after the UEFI boot services have been exited, and must operate on the same hardware as the ACPI TAD before UEFI boot services are exited.
<code>GetTime()</code> and <code>SetTime()</code> MUST be appropriately described in the <code>EFI_RT_PROPERTIES_TABLE</code>.</td></tr>
<tr><td><code>URT_030</code></td><td>The UEFI <code>ResetSystem()</code> runtime service MUST be implemented.</td></tr>
</table>

ID#	Rule
The OS MUST call the `ResetSystem()` runtime service call to reset or shutdown the system, preferring this to SBI, ACPI or other system-specific mechanisms. This allows for systems to perform any required system tasks on the way out (e.g. servicing `UpdateCapsule()` or persisting non-volatile variables in some systems).	
`URT_040`	The non-volatile UEFI variables MUST persist across calls to the `ResetSystem()` runtime service call.
This rule is included in this specification to address a common mistake in implementing the UEFI requirements for non-volatile variables, even though it may appear redundant with the existing UEFI specification.	
`URT_050`	UEFI runtime services MUST be able to update the UEFI variables directly without the aid of an OS.
UEFI variables are normally saved in a dedicated storage which is not directly accessible by the operating system.	

5.3. BRS-I Security Requirements

ID#	Rule
`USEC_010`	Systems implementing a TPM MUST implement the *TCG EFI Protocol Specification* [18].
`USEC_020`	Systems with UEFI secure boot MUST support a minimum of 128 KiB of non-volatile storage for UEFI variables.
`USEC_030`	For systems with UEFI secure boot, the maximum supported variable size MUST be at least 64 KiB.
`USEC_040`	For systems with UEFI secure boot, the `db` signature database variable (`EFI_IMAGE_SECURITY_DATABASE`) MUST be created with `EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS`, to prevent rollback attacks.
`USEC_050`	For systems with UEFI secure boot, the `dbx` signature database variable (`EFI_IMAGE_SECURITY_DATABASE1`) MUST be created with `EFI_VARIABLE_TIME_BASED_AUTHENTICATED_WRITE_ACCESS`, to prevent rollback attacks.

See additional requirements for UEFI runtime services.

5.4. BRS-I Firmware Update

ID#	Rule
`UFU_010`	Systems with in-band firmware updates MUST do so either via `UpdateCapsule()` UEFI runtime service ([10] Section 8.5.3) or via *Delivery of Capsules via file on Mass Storage Device* ([10] Section 8.5.5).
In-band means the firmware running on a hart updates itself.	

ID#	Rule
`UFU_020`	Systems implementing in-band firmware updates via `UpdateCapsule()` MUST accept updates in the *Firmware Management Protocol Data Capsule Structure* format as described in *Delivering Capsules Containing Updates to Firmware Management Protocol* [10] (Section 23.3).
`UFU_030`	Systems implementing in-band firmware updates via `UpdateCapsule()` MUST provide an ESRT [10] (Section 23.4) describing every firmware image that is updated in-band.
`UFU_040`	Systems implementing in-band firmware updates via `UpdateCapsule()` MAY return `EFI_UNSUPPORTED`, when called after the UEFI boot services have been exited.
See additional guidance.	

Chapter 6. BRS-I ACPI Requirements

The *Advanced Configuration and Power Interface Specification* provides the OS-centric view of system configuration, various hardware resources, events and power management.

This section defines the BRS-I mandatory and optional ACPI requirements on top of existing ACPI [3] and UEFI [10] specification requirements. Additional non-normative guidance may be found in the firmware implementation guidance section.

All content in this section is optional and recommended for BRS-B.

ID#	Rule
`ACPI_010`	Be 64-bits clean. • RSDT MUST NOT be implemented, with `RsdtAddress` in RSDP set to 0. • 32-bit address fields MUST be 0.
See additional guidance.	
`ACPI_020`	MUST implement the hardware-reduced ACPI mode (no FACS table).
See additional guidance.	
`ACPI_030`	The Processor Properties Table (PPTT) MUST be implemented, even on systems with a simple hart topology.
`ACPI_040`	The PCI Memory-mapped Configuration Space (MCFG) table MUST NOT be present if it violates [17].
Only compatible PCIe segments, exposed via ECAM (Enhanced Configuration Access Mechanism), may be described in the MCFG. The MCFG MUST NOT require vendor-specific OS support. See PCI Services ([3], Section 4) for more ACPI requirements relating to PCIe support. See additional guidance.	
`ACPI_050`	A Serial Port Console Redirection Table [19] MUST be present on systems, where the graphics hardware is not present or not made available to an OS loader via the standard `UEFI EFI_GRAPHICS_OUTPUT_PROTOCOL` interface.
In these cases, the table provides essential configuration for an early OS boot console.	
`ACPI_060`	An SPCR table, if present, MUST meet the following requirements: • Revision 4 or later of SPCR. • For NS16550-compatible UARTs: 　◦ Use `Interface Type` 0x12 (16550-compatible with parameters defined in Generic Address Structure). 　◦ There MUST be a matching AML device object with `_HID` (Hardware ID) or `_CID` (Compatible ID) `RSCV0003`.
See additional guidance.	

6.1. BRS-I ACPI Methods and Objects

This section lists additional requirements for ACPI methods and objects.

See additional guidance.

<table>
<tr><th>ID#</th><th>Rule</th></tr>
<tr><td>AML_010</td><td>The Current Resource Setting (_CRS) device method for a PCIe Root Complex SHOULD NOT return any descriptors for I/O ranges (such as created by ASL macros WordIO, DWordIO, QWordIO, IO, FixedIO, or ExtendedIO).</td></tr>
<tr><td colspan="2">Legacy PCI I/O BARs are uncommon in modern PCIe devices and support for PCI I/O space may complicate configuration of PCIe Root Complex hardware in a compliant manner.</td></tr>
<tr><td>AML_020</td><td>The Possible Resource Settings (_PRS) and Set Resource Settings (_SRS) device method SHOULD NOT be implemented.</td></tr>
<tr><td colspan="2">ACPI resource descriptors are typically used to describe devices with fixed I/O regions that do not change. Flexible resource assignment is not supported by most modern ACPI OSes.</td></tr>
<tr><td>AML_030</td><td>Per-hart device objects MUST be defined under _SB (System Bus) namespace and not in the deprecated _PR (Processors) namespace.</td></tr>
<tr><td>AML_040</td><td>Systems supporting OS-directed hart performance control and power management MUST expose these via Collaborative Processor Performance Control (CPPC, [3] Section 8.4.6).</td></tr>
<tr><td>AML_050</td><td>Processor idle states MUST be described using Low Power Idle (_LPI, [3] Section 8.4.3).</td></tr>
<tr><td>AML_060</td><td>Systems with a Real-Time Clock on an OS-managed bus (e.g. I2C, subject to arbitration issues due to access to the bus by the OS) MUST implement the Time and Alarm Device (TAD) with functioning _GRT and _SRT methods, and the _GCP method returning bit 2 set (i.e. get/set real time features implemented).</td></tr>
<tr><td colspan="2">Also see URT_020.</td></tr>
<tr><td>AML_070</td><td>Systems implementing a TAD MUST be functional without additional system-specific OS drivers.</td></tr>
<tr><td colspan="2">In situations where the Time and Alarm Device (TAD) depends on a vendor-specific OS driver for correct function (SPI, I2C, etc), the TAD MUST be functional if the OS driver is not loaded. That is, when a dependent driver is loaded, an AML method switches further accesses to go through the driver-backed OperationRegion.</td></tr>
<tr><td>AML_080</td><td>PLIC and APLIC device objects MUST support the Global System Interrupt Base (_GSB, [3] Section 6.2.7) object. See additional guidance.</td></tr>
<tr><td>AML_090</td><td>UART device objects with ID RSCV0003 MUST implement Properties for UART Devices.</td></tr>
<tr><td>AML_100</td><td>PLIC/APLIC namespace devices MUST be present in the ACPI namespace whenever corresponding MADT entries are present. See RVI ACPI IDs.</td></tr>
<tr><td colspan="2">Also see AML_080 and additional guidance.</td></tr>
</table>

6.2. RVI-specific ACPI IDs

ACPI ID is used in the `_HID` (Hardware ID), `_CID` (Compatible ID) or `_SUB` (Subsystem ID) objects as described in the ACPI Specification for devices, that do not have a standard enumeration mechanism. The ACPI ID consists of two parts: a vendor identifier followed by a product identifier.

Vendor IDs consist of 4 characters, each character being either an uppercase letter (A-Z) or a numeral (0-9). The vendor ID SHOULD be unique across the Industry and registered by the UEFI forum. For RVI standard devices, `RSCV` is the vendor ID registered. Vendor-specific devices can use an appropriate vendor ID registered for the manufacturer.

Product IDs are always four-character hexadecimal numbers (0-9 and A-F). The device manufacturer is responsible for assigning this identifier to each product model.

This document contains the canonical list of ACPI IDs for the namespace devices that adhere to the RVI specifications. The RVI task groups may make pull requests against this repository to request the allocation of ACPI ID for any new device.

ACPI ID	Device
`RSCV0001`	RISC-V Platform-Level Interrupt Controller (PLIC)
`RSCV0002`	RISC-V Advanced Platform-Level Interrupt Controller (APLIC)
`RSCV0003`	NS16550 UART compatible with an SPCR definition using `Interface Type` 0x12
`RSCV0004`	RISC-V IOMMU implemented as a platform device
`RSCV0005`	RISC-V SBI Message Proxy (MPXY) Mailbox Controller
`RSCV0006`	RISC-V RPMI System MSI Interrupt Controller
Also see Section 6.4.	

6.3. RVI-specific ACPI Device Properties

This section is used to define the `_DSD` device properties [20] in the `rscv-` namespace.

Where explicit values are provided in a property definition, only these values must be used. System behavior with any other values is undefined.

Property	Type	Description
Currently, there are no properties defined in the `rscv-` namespace. Request for new property names in the `rscv-` namespace should be made as a git pull request to this table.		

6.4. ACPI Device Properties for UART Devices

Generic 16550-compatible UART devices can have device properties in the global name space since Operating Systems are already using them.

Property	Type	Description
`clock-frequency`	Integer	Clock feeding the IP block in Hz.
A value of zero will preclude the ability to set the baud rate, or to configure a disabled device.		
`reg-offset`	Integer	Offset to apply to the register map base address from the start of the registers.
`reg-shift`	Integer	Quantity to shift the register offsets by.
`reg-io-width`	Integer	The size (in bytes) of the register accesses that should be performed on the device.
1, 2, 4 or 8.		
`fifo-size`	Integer	The FIFO size (in bytes).

Chapter 7. BRS-I SMBIOS Requirements

The *System Management BIOS (SMBIOS) Reference Specification* defines a standard format for presenting management information about an implementation, mostly focusing on hardware components.

This section defines the BRS-I mandatory and optional SMBIOS requirements on top of existing [9] specification requirements. Additional non-normative guidance may be found in the firmware implementation guidance section.

All content in this section is optional and recommended for BRS-B.

The structures and fields in this section are defined in a manner consistent with the DMTF specification language ([9]).

ID#	Rule
SMBIOS_010	A Baseboard/Module Information (Type 02) structure SHOULD be implemented.
This relaxes the SMBIOS specification requirement.	
SMBIOS_020	Processor Information (Type 04) structures, meeting the additional Section 7.1 clarifications, MUST be implemented.
This supersedes the RISC-V specific language in the SMBIOS specification ([9], Section 7.5.3.5).	
SMBIOS_030	Port Connector Information (Type 08) structures SHOULD be implemented.
SMBIOS_040	BIOS Language Information (Type 13) structures SHOULD be implemented.
SMBIOS_050	An IPMI Device Information (Type 38) structure MUST be implemented, when an IPMIv1.0 host interface is present.
SMBIOS_060	System Power Supply (Type 39) structures SHOULD be implemented.
SMBIOS_070	Onboard Devices Extended Information (Type 41) structures SHOULD be implemented.
SMBIOS_080	A Redfish Host Interface (Type 42) structure MUST be implemented, when a Redfish host interface is present.
SMBIOS_090	A TPM Device (Type 43) structure MUST be implemented, when a TPM is present.
SMBIOS_100	Processor Additional Information (Type 44) structures MUST be implemented.
See the structure definitions below.	
SMBIOS_110	Firmware Inventory Information (Type 45) structures SHOULD be implemented.

7.1. Type 04 Processor Information

The information in this section supersedes the definitions in ([9], Section 7.5.3.4).

A processor is a grouping of harts in a physical package. In modern designs this MAY mean an SoC.

For RISC-V class CPUs, the `Processor ID` field contains two `DWORD`-formatted values describing the overall physical processor package vendor and version. For some implementations this may also be known as the SoC ID. The first `DWORD` (offsets 08h-0Bh) is the JEP-106 code for the vendor, where bits 6:0 is the ID without the parity and bits 31:7 represent the number of continuation codes. The second `DWORD` (offsets 0Ch-0Fh) reflects vendor-specific part versioning.

For hart-specific vendor and revision information, please see Section 7.2.

7.2. Type 44 Processor-Specific Data

The processor-specific data structure fields are defined to follow the standard Processor-Specific Block fields ([9], Section 7.45.1).

The structure is valid for processors declared with `Processor Type` 07h (64-bit RISC-V) only.

A Type 44 structure needs to be provided for every hart meeting Chapter 3 requirements.

Offset	Version	Name	Length	Value	Description
00h	0100h	`Revision`	`WORD`	Varies	See Section 7.3.
02h	0100h	`Hart ID`	`QWORD`	Varies	The ID of this RISC-V hart.
0Ah	0100h	`Machine Vendor ID`	`QWORD`	Varies	The vendor ID of this RISC-V hart.
12h	0100h	`Machine Architecture ID`	`QWORD`	Varies	Base microarchitecture of the hart. Value of 0 is possible to indicate the field is not implemented. The combination of `Machine Architecture ID` and `Machine Vendor ID` should uniquely identify the type of hart microarchitecture that is implemented.
1Ah	0100h	`Machine Implementation ID`	`QWORD`	Varies	Unique encoding of the version of the processor implementation.

7.3. Processor-Specific Data Structure Versioning

The processor-specific data structure begins with a revision field to allow for future extensibility in a backwards-compatible manner.

The minor revision is to be incremented anytime new fields are added in a backwards-compatible manner. The major revision is to be incremented on backwards-incompatible changes.

Version	Bits 15:8+ Major revision	Bits 7:0+ Minor revision	Combined	Description
v1.0	01h	00h	0100h	First BRS-defined definition

Chapter 8. Firmware Implementation Guidance

The guidance section is non-normative, and covers certain implementation choices, suggestions, historical context, etc.

8.1. Recipes Guidance

8.1.1. BRS-I Recipe Guidance

Systems compliant to BRS-I can successfully boot an existing generic operating system image without system-specific customizations, yet this might result in an unoptimized experience and non-functioning I/O devices until further software updates are activated.

The best analogy would be a typical Intel Architecture motherboard from the early 2000s: you could install an OS on it, but the built-in graphics might be low-resolution and the sound, built-in network port or power management might not work out of the box. You could subsequently load the right drivers from the media coming with the board or fetch newest ones using a well-supported network adapter.

8.2. UEFI Implementation Guidance

UEFI implementations run in 64-bit S-Mode, VS-Mode or HS-Mode, depending on whether virtualization is supported or used.

8.2.1. Privilege Levels

Different portions of system firmware might target a specific privilege level. In contrast, UEFI drivers, OS loaders and pre-boot applications need to operate in supervisor mode (either (V)S-Mode or HS-Mode), because they are UEFI-compliant executable images.

As an implementation choice, a UEFI firmware implementation may start execution in M-Mode. However it must switch to supervisor mode as part of initializing boot and runtime services for UEFI drivers and applications.

8.2.2. Firmware Update

`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.

8.2.3. PCIe

Every implementation of the `EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL` provides the correct `Address`

`Translation Offset` field to translate between the hart MMIO and bus addresses.

`EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_CONFIGURATION` structures report resources produced by the PCIe root bridges, not resources consumed by their register maps. In the cases where there are unpopulated PCIe slots behind a root bridge, `EFI_PCI_ROOT_BRIDGE_IO_PROTOCOL_CONFIGURATION` reports valid resources assigned (e.g. for hot plug), or reports no resources assigned.

Firmware MUST always initialize PCIe root complexes, even if booting from non-PCIe devices, and should not assume the OS knows how to configure root complex hardware (including, for example, inbound and outbound address translation windows). In fact, ECAM-compatible PCIe segments are assumed by operating systems to just work as per their hardware descriptions in ACPI and DT. Furthermore, firmware MUST perform BAR resource assignment, bridge bus number and window assignments and other reasonable device setting configuration (e.g. Max Payload Size) and not assume operating systems to be capable of full PCIe resource configuration, or to expect full reconfiguration to be necessary.

8.2.4. UEFI Runtime Services

Systems without an RTC and with an equivalent alternate source for current time, that is not trivially accessible from a UEFI implementation after the UEFI boot services are shut down (e.g. network time), can implement `GetTime()` using the `time` CSR. The time, of course, needs to be synchronized before the boot services are shut down.

8.3. ACPI Implementation Guidance

ACPI information is structured as tables with the address of the root of these tables known as Root System Description Table (RSDP) passed to the OS via a `EFI_ACPI_20_TABLE_GUID` configuration table in the UEFI firmware. The Operating System uses this address to locate all other ACPI tables.

Certain implementations may make use of the *RISC-V Functional Fixed Hardware Specification* [21].

8.3.1. 64-bits Clean

ACPI started as a specification for 32-bit systems, so certain tables with physical address pointers (e.g. RSDP, FADT) allow for reporting either 32-bit or 64-bit values using different fields. For the sake of simplicity and consistency, the BRS disallows the use 32-bit address fields in such structures and disallows the use of 32-bit only structures (thus, RSDT must not be implemented, as the XSDT is a direct replacement). Thus, the ACPI tables are allowed to be located in any part of the physical address space.

8.3.2. Hardware-Reduced ACPI

Compliant RISC-V systems only implement the hardware-reduced ACPI model [3] (Section 4.1). This means the hardware portion of [3] (Section 4) is not required or supported. All functionality is instead provided through equivalent software-defined interfaces and the complexity in supporting ACPI is reduced.

8.3.3. Table Guidance

Table 4 summarizes the minimum set of structures that must exist to support basic booting of RISC-V system with ACPI support. Table 5 lists additional possible ACPI tables based on the optional features that can be supported. The latter is not meant to be exhaustive and mostly focuses on tables that have specific guidance or that are expected to be frequently implemented.

Table 4. **Minimum required ACPI System Description Tables**

ACPI Table	ACPI Section	Note
Root System Description Pointer (RSDP)	5.2.5	See high-level requirements.
Extended System Description Table (XSDT)	5.2.8	Contains pointers to other tables.
Fixed ACPI Description Table (FADT)	5.2.9	See `ACPI_020`, Section 8.3.2 and the notes below.
Differentiated System Description Table (DSDT)	5.2.11.1	See Section 6.1 and the notes below.
Multiple APIC Description Table (MADT)	5.2.12	See the notes below
RISC-V Hart Capabilities Table (RHCT)	New	Communicates information about certain capabilities like ISA string, cache and MMU info.
Processor Properties Topology Table (PPTT)	5.2.29	See `ACPI_030`

Table 5. **Additional ACPI System Description Tables based on feature support.**

ACPI Table	ACPI Section	Note
Memory-mapped Configuration space (MCFG)	[17]	See `ACPI_040` and the notes below
Secondary System Description Table (SSDT)	5.2.11.2	See Section 6.1 and the notes below.
Serial Port Console Redirection (SPCR)	[19]	See `ACPI_060` and the notes below
ACPI Table for TPM 2.0 (TPM2)	[22]	If the system supports TPM 2.0
System Resource Affinity Table (SRAT)	5.2.16	If the system supports NUMA
System Locality Information Table (SLIT)	5.2.17	If the system supports NUMA
Boot Error Record Table (BERT)	18.3.1	If APEI is supported

ACPI Table	ACPI Section	Note
Error Injection Table (EINJ)	18.6.1	If APEI is supported
Error Record Serialization Table (ERST)	18.5	If APEI is supported
Hardware Error Source Table (HEST)	18.3.2	If APEI is supported
RISC-V IO Mapping Table (RIMT)	New	If the system supports IOMMU

8.3.4. DSDT and SSDTs

The ACPI name space describes devices which cannot be enumerated by any other standard ways. These typically include SoC embedded memory-mapped I/O devices, such as UARTs, PCIe or CXL root complexes, GPIO controllers, etc.

It’s an implementation choice if the ACPI name space is defined solely with a DSDT or with any additional SSDTs. For example, a UEFI implementation may choose to use SSDTs to:

- describe devices that vary across SoC SKUs, revisions or variants.
- describe devices, where the backing AML is generated or patched at boot time.

8.3.5. FADT

[3] (Section 5.2.9) provides guidance on filling the Fixed ACPI Description Table for HW-reduced ACPI.

Don’t forget to select an appropriate Preferred PM Profile.

8.3.6. MADT

RINTC (per-hart) structures are mandatory. Depending on the interrupt controller implemented by the system, the MADT will also contain either PLIC or APLIC structures.

Entry ordering can be correlated with initialization order by an OS, but should not be taken to reflect affinity in resource sharing, e.g. sockets, caches, etc. RINTC hart ID and ACPI Processor UID should not be decoded in a system-specific manner to divine CPU topology. The PPTT Processor Properties Topology Table (PPTT) is to be used to describe affinities.

8.3.7. PLIC/APLIC Namespace Devices

Here’s an example of an ASL excerpt satisfying `AML_100` and `AML_080` requirements.

```
Scope (\_SB)
{
   Device (IC00)
   {
       Name (_HID, "RSCV0001") // _HID: Hardware ID
       Name (_UID, Zero)  // _UID: Unique ID
       Method(_GSB) {
```

```
            Return (0x10) // Global System Interrupt Base for this PLIC starts at 16
        }
        Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource Settings
        {
            Memory32Fixed (ReadWrite,
              0x0C000000,         // Address Base.
              0x00220000,         // Address Length
              )
        })
    }
    Device (DEV1)
    {
      ...
      Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource Settings
      {
          Memory32Fixed (ReadWrite,
            0x10010000,         // Address Base.
            0x00010000,         // Address Length
            )
          Interrupt (ResourceConsumer, Level, ActiveHigh, Exclusive, ,,)
          {
            0x10,
          }
      })
    }
}
```

8.3.8. PCIe

On some architectures, it became an industry accepted norm to describe PCIe implementations not compliant to the *PCI Firmware Specification* [17] using specification-defined ACPI tables and objects. RISC-V systems compliant to the BRS must only expose ECAM-compatible implementations using the MCFG and the standard AML Hardware ID (`_HID`) `PNP0A08` and Compatible ID (`_CID`) `PNP0A03`, and must not rely on ACPI table header information or other out-of-band means of detecting quirked behavior.

Some minor incompatibilities, such as incorrect CFG0 filtering, broken BARs/capabilities for RCs, embedded switches/bridges or embedded endpoints can be handled by emulating ECAM accesses in privileged firmware (e.g. M-mode) or similar facilities (e.g. a hypervisor).

Non-compliant implementations must be exposed using vendor-specific mechanisms (e.g. AML object with custom `_HID`, custom vendor-specific ACPI table if necessary). In cases where such PCIe implementations are only used to expose a fixed non-removable device (e.g. USB host controller or NVMe), the device could be exposed via a DSDT/SSDT MMIO device object without making the OS aware of the underlying PCIe connection.

8.3.9. SPCR

Early serial console can be implemented using either an NS16550 UART (SPCR `Interface Type` 0x12), a PL011 UART (SPCR `Interface Type` 0x03), or an SBI console (SPCR `Interface Type` 0x15). When SPCR describes SBI console, the OS must use the SBI Probe extension (`FID #3`) to detect the Debug Console Extension (`DBCN`).

The new `Precise Baud Rate` field, introduced in [19] rev. 4, allows describing rates faster than 115200 baud for NS16550-compatible UARTS.

Hardware not capable of interrupt-driven operation and SBI console should be described with `Interrupt Type` of 0 and `Global System Interrupt` of 0.

8.4. SMBIOS Implementation Guidance

Note the DMTF requirements on the 64-bit SMBIOS 3.0 entry point ([9] Section 5.2.2), and the conformance guidelines ([9] Annex A).

8.4.1. Type 44 Processor-Specific Data

The `Machine Vendor ID`, `Machine Architecture ID`, and `Machine Implementation ID` fields typically reflect the `mvendorid`, `marchid` and `mimpid` CSRs respectively.

Bibliography

[1] “Key words for use in RFCs to Indicate Requirement Levels.” [Online]. Available: datatracker.ietf.org/doc/html/rfc2119.

[2] “PCI Express® Base Specification Revision 6.0.” [Online]. Available: pcisig.com/pci-express-6.0-specification.

[3] “Advanced Configuration and Power Interface Specification 6.6.” [Online]. Available: uefi.org/specifications.

[4] “DeviceTree.” [Online]. Available: www.devicetree.org/.

[5] “Embedded Base Boot Requirements Specification 2.1.0.” [Online]. Available: github.com/ARM-software/ebbr/releases/download/v2.1.0/ebbr-v2.1.0.pdf.

[6] “RISC-V Profile.” [Online]. Available: github.com/riscv/riscv-profiles.

[7] “RISC-V Platform Management Interface Specification.” [Online]. Available: github.com/riscv-non-isa/riscv-rpmi.

[8] “RISC-V Supervisor Binary Interface Specification.” [Online]. Available: github.com/riscv-non-isa/riscv-sbi-doc.

[9] “System Management BIOS (SMBIOS) Reference Specification 3.7.0.” [Online]. Available: www.dmtf.org/standards/smbios.

[10] “Unified Extensible Firmware Interface Specification 2.11.” [Online]. Available: uefi.org/specifications.

[11] “RISC-V ‘stimecmp / vstimecmp’ Extension.” 2021, [Online]. Available: github.com/riscv/riscv-time-compare.

[12] “The RISC-V Advanced Interrupt Architecture.” 2023, [Online]. Available: github.com/riscv/riscv-aia.

[13] “RISC-V Indirect CSR Access (Smcsrind/Sscsrind).” 2023, [Online]. Available: github.com/riscv/riscv-indirect-csr-access.

[14] “RISC-V Supervisor Counter Delegation Specification (Smcdeleg/Ssccfg).” 2024, [Online]. Available: github.com/riscv/riscv-smcdeleg-ssccfg.

[15] “UEFI memory mitigations.” [Online]. Available: learn.microsoft.com/en-us/windows-hardware/drivers/bringup/uefi-ca-memory-mitigation-requirements.

[16] “UEFI Platform Initialization Specification 1.9.” [Online]. Available: uefi.org/specifications.

[17] “PCI Firmware Specification Revision 3.3.” [Online]. Available: members.pcisig.com/wg/PCI-SIG/document/folder/862.

[18] “TCG EFI Platform Specification.” [Online]. Available: trustedcomputinggroup.org/resource/tcg-

efi-platform-specification/.

[19] “Serial Port Console Redirection Table (SPCR).” [Online]. Available: learn.microsoft.com/en-us/windows-hardware/drivers/serports/serial-port-console-redirection-table.

[20] “_DSD (Device Specific Data) Implementation Guide.” [Online]. Available: github.com/UEFI/DSD-Guide/blob/main/dsd-guide.pdf.

[21] “RISC-V Functional Fixed Hardware Specification.” [Online]. Available: github.com/riscv-non-isa/riscv-acpi-ffh.

[22] “TCG ACPI Specification.” [Online]. Available: trustedcomputinggroup.org/resource/tcg-acpi-specification/.