Arm® System Control and Management Interface 4.0 alp0

Platform Design Document

Non-confidential

ALPHA0

This is an engineering draft of the specification. It is meant to obtain feedback from Arm partners and internally within Arm. It is subject to change based on this feedback. Changes introduced in this version can be withdrawn, or modified in a manner which breaks backward compatibility in subsequent revisions.

Document number: DEN0056F

Release information

The following table lists the changes that are made to this document:

Date	Issue	Confidentiality	Change
May 2017	Issue A	Non-confidential	Version 1.0, first external release
July 2019	Issue B	Non-confidential	Version 2.0. 1. Removed reference to specific document versions in References. 2. Replaced PSCA with the correct acronym (PCSA) for Power Control System Architecture in Introduction. 3. Added clarifications to SCMI status codes NOT_FOUND and NOT_SUPPORTED in SCMI status codes. 4. Added clarifications on OSPM view in System power management protocol background. 5. Added more context to the OUT_OF_RANGE and BUSY status codes. 6. Added guidance on usage of ACPI PCC channels for SCMI transport. 7. Added clarifying note on power costs of performance domains. 8. Added FastChannel support. 9. Added Power Domain Management pre-notification support. 10. Added Agent-specific Resource Isolation capability as a part of Base protocol. 11. Add agent_id self-discovery. 12. Added notes on agent-id management. 13. Replaced SCMI overview diagram. 14. Cleaned up description/grammar and typos at multiple places. 15. Extended System Power Management Protocol notifier to support Virtualized system implementations. 16. Added Reset Management Protocol. 17. Renamed Mailbox Transport to more appropriate Shared Memory based Transport and made changes to allow SMC/HVC based doorbells. 18. Added guidance on usage of ACPI PCC channels for SCMI transport. 19. Added more context to the OUT_OF_RANGE and BUSY status codes. 20. Added clarifications to SCMI status codes NOT_FOUND and NOT_SUPPORTED. 21. Added support for notifications to agents on performance level change events triggered by external factors. 22. Remove requirement for Statistics Regions to be reset after system suspend.

Date	Issue	Confidentiality	Change
Nov. 2020	Issue C	Non-confidential	Version 3.0. 1. Fixed typographical errors and broken links. 2. Add multi-axis sensor support. 3. Allow reporting sensor resolution and configuring sensor update intervals, timestamps, and sensor state. 4. Add notification support for continuous sensor sampling. 5. Sensor value fields changed from uint32 to int32. 6. Add support for SENSOR_CONFIG_GET/SET and move sensor_update_interval field reporting. 7. Clarify usage of trip points for multi-axis sensors. 8. Change Sensor protocol version to 2.0. 9. Simplify sensor shared memory region by specifying Sensor Value Data Entry Table explicitly. 10. Add guidance in Message format to allow sending only the last notification in case of quick transitions for the same event. This replaces specific guidance in all notification descriptions. 11. Provide guidance of minimum message size to be supported by a transport in Transport. 12. Remove 32-bit restriction for shared memory identifier for SMC/HVC based doorbells. 13. Broaden scope of COMMS_ERROR usage. 14. Fix pseudocode in BASE_DISCOVER_LIST_PROTOCOLS. 15. Clarify Sustained Performance Level in Performance Management Protocol to include external constraints. 16. Clarify that performance level power cost is per AP only when the domain includes APs. Also clarify that reporting power cost is optional. 17. Allow performance level change notifications to be sent to the agent requesting a level change to cater for the case when PERFORMANCE_LEVEL_SET returns asynchronously. 18. Introduce Voltage Domain Protocol. 19. Add agent_id field to RESET_ISSUED notification. 20. Change Reset domain protocol version to 2.0. 21. Specify that reset domain identifiers should be sequential and start from 0. 22. Specify that sensor identifiers should be sequential and start from 0. 23. Clarify CLOCK_DESCRIBE_RATES command to return only one segment in case a triplet is returned. 24. Mandate 64-bit alignment for statistics shared memory regions. 25. Add Match Sequence to Statistics Shared Memory Region to detect race conditions between write accesses by the platform and read accesses by the agent. 26. Change revision field of statistics tables in all protocols to 1.0.

Date	Issue	Confidentiality	Change
			27. Add POWER_STATE_CHANGE_REQUESTED notification attribute bit in POWER_DOMAIN_ATTRIBUTES command. 28. Change Power Domain Protocol version to 2.1. 29. Add a clarificatory phrase in all command returns to allow a generic range of return types. 30. Update SCMI Overview diagram with Voltage Domain Protocol.
Mar 2022	Issue D	Non-Confidential	Version 3.1. 1. Editorial fixes. 2. Update Figure 1 – SCMI Overview. 3. Introducing Progressive Terminology - Completer and Requester. 4. Add Guidance for systems implementing Arm RME, in System Control and Management Interface structure. 5. Power Domain Management Protocol: a. Introduce extended domain name. b. Change Protocol version to 3.0. 6. System Power Management Protocol: a. Introduce 'timeout' field in SYSTEM_POWER_STATE_NOTIFIER. b. Change Protocol version to 2.0. 7. Performance Domain Management Protocol a. Allow PERFORMANCE_LIMITS_SET command to modify either the max. or the min. limit. b. Introduce extended domain name. c. Allow power cost in microwatts. d. Change Protocol version to 3.0. 8. Clock management Protocol: a. Introduce extended domain name. b. Introduce Clock Rate change (pre and post) notification support. c. Add clock_enable_delay field to CLOCK_ATTRIBUTES. d. Change Protocol version to 2.0. 9. Sensor Management Protocol: a. Introduce extended sensor and axis name. b. Change Protocol version to 3.0. 10. Reset Domain Management Protocol: a. Introduce extended domain name. b. Change Protocol version to 3.0. 11. Voltage Domain Management Protocol: a. Add support for async setting of voltage levels and VOLTAGE_LEVEL_SET_COMPLETE delayed response. b. Introduce extended domain name. c. Change Protocol version to 2.0. 12. Introduce Power Capping and Monitoring Protocol. 13. Add reference to Virtio-SCMI transport.

Date	Issue	Confidentiality	Change
Mar 2024	Issue E	Non-Confidential	Version 3.2 1. Editorial fixes. 2. Update Figure 1 – SCMI Overview. 3. Add clarification around reference counting and disallow intra-agent reference counting. Add clarification around physical vs virtual resource views for shared resources. 4. Add IN_USE error code for usage with Pin control protocol. 5. Clarify in Power domain management protocol background guidance around shared power domain handling. 6. Clarify notification behavior in response to system power state requests in System power management protocol background. 7. Add NEGOTIATE_PROTOCOL_VERSION command to all protocols and increment version of the protocol. 8. Performance domain management protocol. a. Add support for hardware which works on performance indices, instead of levels. Introduce 'Level Indexing Mode'. b. Provide guidance around Level Indexing Mode when using ACPI in ACPI-based Transport. c. Change protocol version to 4.0. 9. Clock Management Protocol a. Add note for usage guidance in relevant commands. b. Add guidance for shared clocks handling. c. Add support for discovering and changing parent clocks. d. Add support for configuring extended and OEM configuration type and values. e. Introduce new mandatory command CLOCK_CONFIG_GET. f. Introduce restricted clocks and a new optional command CLOCK_GET_PERMISSIONS. g. Change protocol version to 3.0. 10. Add a note for security guidance in Sensor protocol. 11. Add clarification around shared voltage domains 12. Add clarification around shared reset domains. 13. Power Capping and Monitoring Protocol a. Add Note for security guidance. b. Clarify that power cap is disabled on a domain when power cap is set to 0. c. Clarify that min_power_cap and max_power_cap attributes returned by POWERCAP_DOMAIN_ATTRIBUTES cannot be 0. d. Change Protocol Version to 2.0. 14. Introduce Pin Control Protocol. 15. Clarify definition of length field in Shared memory area layout.

Date	Issue	Confidentiality	Change
May 2025	Issue F	Non-Confidential	Version 4.0 ALP0 1. Editorial fixes and clarifications. 2. Update Figure 1 – SCMI Overview. 3. Add PARTIAL_ERROR error code for usage with System Telemetry protocol. 4. Mark statistics shared memory regions for deprecation in Power Domain, Performance and Sensor Protocol. 5. Update the Sensor Type Enumerations table with new values. 6. Performance Domain Management Protocol a. Add support for Quality of Service, QoS-only domains and parent domains for QoS. b. Add definition of Guaranteed Performance and clarify meaning of Sustained Performance. c. Change Protocol Version to 4.1. 7. Power Capping and Monitoring Protocol a. Add support for Concurrent Power Capping. b. Separate averaging intervals into measurement and concurrent averaging intervals. c. Change Protocol Version to 3.0. 8. Introduce System Telemetry Protocol. 9. Add clarification around message communications flow. 10. Add link to MPAM Firmware-backed Profile Specification with Protocol ID 0x1A. 11. Clarify that the sync command does not necessarily block the channel and that agents can run independently of each other.

Arm Non-Confidential Document License (“License”)

This License is a legal agreement between you and Arm Limited (“**Arm**”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License (“**Document**”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

“**Subsidiary**” means any company the majority of whose voting shares is now or hereafter owned or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries (“Licensee”) is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm’s view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

Reference by Arm to any third party’s products or services within this document is not an express or implied approval or endorsement of the use thereof.

THE DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE’S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Any breach of this License by a Subsidiary shall entitle Arm to terminate this License as if you were the party in breach. Any termination of this License shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This License may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this License and any translation, the terms of the English version of this License shall prevail.

The validity, construction and performance of this License shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

Contents

About this document

This document describes an extensible operating system-independent software interface to perform various system control and management tasks, including power and performance management.

References

This document refers to the following documents.

Reference	Document Number	Title
[ACPI]		Advanced Configuration and Power Interface Specification. See https://uefi.org/specifications
[ARM]	DDI 0487	Arm Architecture Reference Manual ARMv8, for ARMv8-A architecture profile.
[ARMTF]		Arm Trusted Firmware. See https://github.com/ARM-software/arm-trusted-firmware.
[CCA]		Arm Confidential Compute Architecture. See https://www.arm.com/why-arm/architecture/security-features/arm-confidential-compute-architecture
[FDT]		Flattened Device Tree. See https://www.devicetree.org
[MPAMFb]	DEN0144	MPAM Firmware-backed Profile Specification. See https://developer.arm.com/documentation/den0144/latest
[PCSA]	DEN0050	Power Control System Architecture Specification.
[PSCI]	DEN0022	Power State Coordination Interface. See https://developer.arm.com/documentation/den0022/latest/
[RME]	DDI0615	The Realm Management Extension (RME), for Armv9-A. See https://developer.arm.com/documentation/ddi0615/latest/
[RMESYS]	DEN0129	Arm Realm Management Extension (RME) System Architecture. See https://developer.arm.com/documentation/den0129/latest/
[SMCCC]	DEN0028	Arm SMC Calling Convention.
[UUID]	RFC 9562	https://datatracker.ietf.org/doc/html/rfc9562
[VIRTIO]		https://github.com/oasis-tcs/virtio-spec

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
ACPI	Advanced Configuration and Power Interface
Agent	Entity that sends commands to the platform using SCMI. For example, the OSPM running on an AP or an on-chip management controller.

AP	Application processor, that is a processor that is running the operating system and applications in the system.
ASL	ACPI Source Language. Interpreted language that is used by the boot firmware to describe methods and data for the Operating System to use to discover and configure system resources. Defined in [ACPI].
Channel	The transport link over which the agent communicates to the platform.
Command	A message that is sent from an agent to the platform.
Completer	An agent in a computing system that responds to and completes a transaction that was initiated by a Requester.
Delayed response	A message that is sent from the platform to an agent to indicate completion of the work that is associated with an asynchronous command.
FastChannel	A FastChannel is a lightweight unidirectional channel that is dedicated to a particular SCMI message for controlling a particular platform resource.
FDT	Flattened Device Tree
Message	An individual communication from an agent to the platform or from the platform to an agent.
MMIO	Memory Mapped IO.
MPAM	Memory System Resource Partitioning And Monitoring
MPAM-Fb	MPAM Firmware-backed
Notification	A message that is sent from the platform to an agent to alert of a change in state.
OSPM	Operating System-directed Power Management. Typically, this acronym refers to the software components of an Operating System that interact with the power management interfaces of the platform.
Platform	The set of system components that interpret the SCMI messages and provide the necessary functionality. An SCP is an example of a platform component that could implement the SCMI messages.
PSCI	Power State Coordination Interface.
Requester	An agent in a computing system that is capable of initiating transactions.
SCMI	System Control and Management Interface, which is described in this specification.
SCP	System Control Processor, see [PCSA].

Feedback

Arm welcomes feedback on its documentation.

If you have any comments or suggestions for additions and improvements create a ticket at https://support.developer.arm.com.

As part of the ticket include:

- The title (Arm® System Control and Management Interface).
- The document ID and version (DEN0056F 4.0 alp0).
- The page numbers to which your comments apply.

- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

—— **Note** ——————————

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

————————————————

Inclusive terminology commitment

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

Previous issues of this document included terms that can be offensive. We have replaced these terms. If you find offensive terms in this document, please contact terms@arm.com.

1 Introduction

This document describes the *System Control and Management Interface* (SCMI), which is a set of operating system-independent software interfaces that are used in system management. SCMI is extensible and currently provides interfaces for:

- Discovery and self-description of the interfaces it supports.
- Power domain management, which is the ability to place a given device or domain into the various power-saving states that it supports.
- Performance management, which is the ability to control the performance of a domain that is composed of compute engines such as *application processors* (APs), GPUs, or other accelerators.
- Clock management, which is the ability to set and inquire rates on platform-managed clocks.
- Sensor management, which is the ability to read sensor data, and be notified of sensor value changes.
- Reset domain management, which is the ability to place a given device or domain into various reset states.
- Voltage domain management, which is the ability to configure and manage the voltage level of a domain that provides voltage supply to a group of components.
- Power capping and monitoring, which is the ability to configure, set power caps and monitor the power consumption of power capping domains.
- Pin control protocol, which is intended for controlling pins or groups of pins, and their configuration.
- System Telemetry protocol, which is intended for gathering system telemetry.

There is a strong trend in the industry to provide microcontrollers in systems to abstract various power, or other system management tasks, away from APs. These controllers usually have similar interfaces, both in terms of the functions that are provided by them, and in terms of how requests are communicated to them. The *Power Control System Architecture* (PCSA) describes how systems using this approach can be built. For detailed information about the PCSA, see [PCSA].

PCSA defines the concept of the *System Control Processor* (SCP), a processor that is used to abstract power and system management tasks from the APs. The SCP can take requests from APs and other system agents. It can coordinate these requests and place components in the platform into appropriate power and performance states. SCMI is particularly relevant to these kinds of systems.

The interface defined by SCMI provides two levels of abstraction:

- **Protocols**
  Each group of related functions is referred to as a protocol. The SCMI interface structure is extensible, and therefore other protocols could be added in the future.
- **Transports**
  The protocols communicate through transports. A transport specification describes how protocol messages are communicated between agents using the interface and the platform components that implement the protocol messages.

The interface is intended to be described in firmware, using either the *Flattened Device Tree* (FDT) or *Advanced Configuration and Power Interface* (ACPI) specification. For more information, see [FDT] and [ACPI]. Because the protocols are intended to be generic, they result in generic kernel code to drive them. However, in the ACPI case, the interface can also be driven from ASL methods.

This document is arranged into the following sections:

- Section 2 provides background into the interface structure.
- Section 3 describes protocols.
- Section 4 describes transports.

2 System Control and Management Interface structure

System Control and Management Interface is intended to allow agents such as an operating system to manage various functions that are provided by the hardware platform it is running on, including power and performance functions. As described in the introduction, SCMI provides two levels of abstraction: protocols and transports.

Some commonly used terms in this specification are described below.

- *Protocols* define individual groups of system control and management messages. A protocol specification describes the messages that it supports.
- *Agent* is used to describe the caller of the System Control & Management Interface.
- *Platform* describes the set of hardware components that interpret the messages and provide the necessary functionality.
- *Resource* is used to describe any component of the hardware platform which can be controlled using SCMI messages.
- *Transport* describes the method by which protocol messages are communicated between agents and the platform. Arm strongly recommends that transports be operating system independent and capable of being virtualized.

It is intended that protocols and transports are developed independently. Protocols are designed to minimize the ability of an agent to exploit a resource to adversely impact the stability, confidentiality, integrity, or security of the system.

A transport might support multiple channels. Each agent that communicates with the platform must have its own set of dedicated channels. In other words, channels cannot be shared between agents. This requirement removes the need for creating locking primitives across agents that are running entirely different software stacks, for example, a management controller and an operating system. In addition, dedicated channels provide a method for the platform to identify which agent is sending a message.

Systems that use Arm TrustZone technology can have both Secure and Non-secure channels. Agents can be in the Secure or Non-secure Security state. A Non-secure channel cannot be used to access or modify Secure platform resources. A Secure channel can only be accessed by an agent in the Secure state.

The Arm Realm Management Extension [RME, RMESYS] introduces two new Security States, Realm and Root, in addition to the existing Secure and Non-Secure states. It also adds two new *Physical Address Spaces* (PASs), Realm and Root, in addition to the existing Secure and Non-secure PAS. A system implementing SCMI can have channels residing in the Root, Secure, and Non-secure PAS. Agents can be in the Root, Secure or Non-secure Security state. Access to resources through a channel in a specific PAS depends on the Security State of the agent making the access in accordance with RME PAS access rules, as specified in [RME].

Table 1 specifies the access rules in the context of SCMI agents, channels, and resources.

—— Note ——————————

SCMI usage by software or firmware running in Realm security state is not anticipated and is outside the scope of this version of the specification.

Table 1: SCMI Channel and Resource Access Rules

Agent Security state	SCMI Resource Assignment and Channel PAS		
	Non-secure	Secure	Root
Non-Secure	Allow	Deny	Deny
Secure	Allow	Allow	Deny

Root	Allow	Allow	Allow

The platform can assign a resource to be controlled from a Root, Secure or Non-secure channel. The resource assignment policy is determined by:

- the Security state of the agent which needs to control the resource, and
- the ability of the agent to exploit the resource to adversely impact the confidentiality, integrity, or security of other agents, or software or firmware resident in other Security states in the system. Specifically, the resource assignment should ensure that the Security guarantees provided by Arm TrustZone and Arm RME are not violated.

As an example, consider a resource which is shared by or which can affect software or firmware resident in Root, Realm, Secure or Non-Secure states. All agents using such a resource should be able to control the resource. In this scenario, the platform must ensure that the resource configuration or state does not adversely impact any entity resident in Root, Realm, Secure or Non-Secure states.

The protocols that are described in this document are intended to be used by power and performance management agents such as an operating system, also referred to as *Operating System-directed Power Management* (OSPM). Typical agents are:

- An OSPM that operates in Non-secure Exception levels.
- Secure-world software that is running on an AP.
- Firmware running in Root Security state in systems implementing Arm CCA.
- A privileged entity like a hypervisor on virtualized systems.
- External entities in the system, such as a management controller in an enterprise system, or a modem in a mobile system.

Resources that are shared between multiple agents should be in a state which meets the requirements of all agents sharing the resource. Any agent that needs to use a shared resource should explicitly make a request for it. The platform may disable a resource if no agent has requested to use that resource.

A reference counting scheme can be used to track requests from different agents to ensure that a shared resource is in the correct state. For example, consider a system with two agents sharing a resource. When the first agent requests for the resource to be enabled, the reference count is incremented to one, and the resource is enabled. When the second agent requests for the resource to be enabled, the reference count is increased to two, but the resource state is left unchanged. The reference count is decremented when an agent requests the resource to be disabled. The resource can be disabled only the reference count is zero. The reference count decrements to zero only when all agents which had previously requested the resource to be enabled have subsequently requested the resource to be disabled. However, the platform should not reference count calls from the same agent. For example, if an agent repeats a command to enable the same resource twice, the platform takes no specific action on the second call and the reference count is not incremented. The specification assumes inter-agent reference counting only and no intra-agent reference count is allowed.

A consequence of reference counting is that a shared resource might not be in the exact state that an agent expects it to be in. For example, an agent might ask for a resource to be disabled. However, the shared resource might not actually be disabled if there are other agents requesting that resource to be kept enabled. An agent can use protocol messages to query the state of a resource. In this case, the platform can choose to provide the actual physical state of the resource which shows that the resource is enabled. Alternatively, the platform could choose to provide an agent specific virtual view of the shared resource, which indicates that the resource is disabled. It is implementation defined whether a physical and virtual view of the resource is provided by the platform to each agent.

Figure 1 below illustrates an example system that implements the SCMI interface. In this example, the platform includes an SCP that handles SCMI commands that are issued from APs. The latter communicates with the SCP through Secure/Root and Non-secure channels. The figure also shows a device that uses SCMI protocols to manage its power

and performance. As described in [PCSA], the SCP coordinates requests from all requesting agents and drives the hardware into appropriate power or performance states.

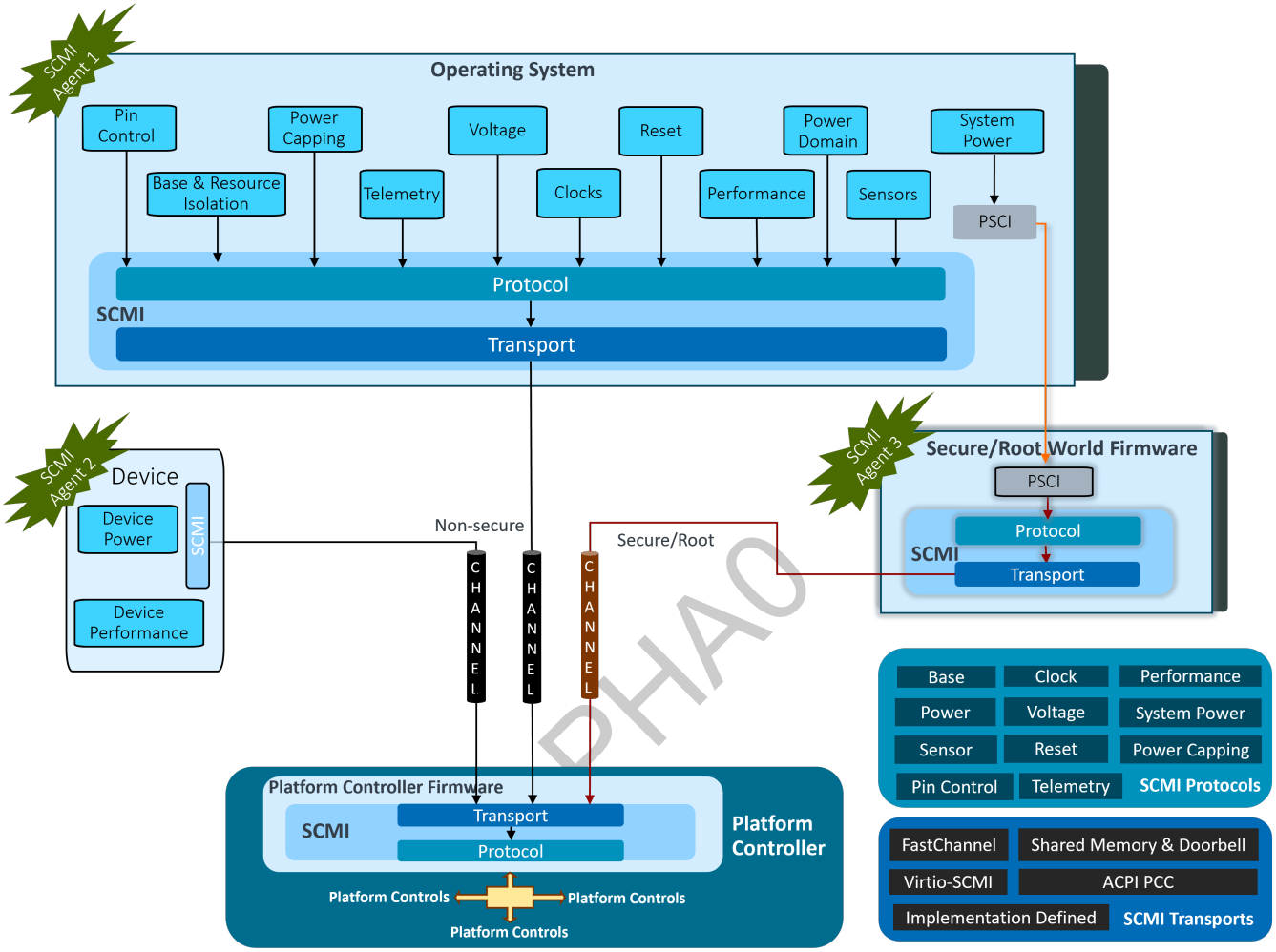

Figure 1: SCMI Overview

3 Protocols

3.1 Protocol structure

As described in Section 2, a protocol is a group of messages. The following sections describe the message flow, the structure of messages, and protocol discovery.

3.1.1 Agents, messages, and channels

The term agent is used to describe components that are clients of the SCMI interface. Agents have the following properties:

- Agents may run a software or firmware stack with different privilege levels.
- Agents are independent from each other. This makes resource sharing, or the ability to write cross-agent locking primitives difficult. For example, one agent might be an operating system running on all APs, and another agent might be firmware running on a manageability controller.
- Agents communicate with the platform independently of each other. For example, the agent representing the firmware running on a management microcontroller can communicate with the platform even when the agent representing the host operating system is not active.

Agents and the platform communicate over transport channels. A channel can be a dedicated SCMI FastChannel or a standard SCMI channel.

A FastChannel is a lightweight unidirectional channel that is dedicated to a single SCMI message type for controlling a specific platform resource, or for getting information about a specific resource. Unlike a standard channel, a FastChannel cannot be used to carry multiple message types, or to explicitly control multiple platform resources. A FastChannel cannot be shared among agents. The absence of multiple message types and their header requirements enables FastChannels to provide a potentially low latency mechanism for an agent to communicate with the platform. However, a FastChannel does not guarantee that the time taken by the platform to complete the requested operation is lower compared to a standard channel. Since FastChannels are protocol and message specific, their behavior is detailed in the respective Protocol sections. Unless explicitly specified, the word ‘channel’ in the rest of the document will always refer to standard SCMI channels.

Figure 2 below describes how agents and the platform communicate over channels. The diagram shows multiple agents communicating with the platform.

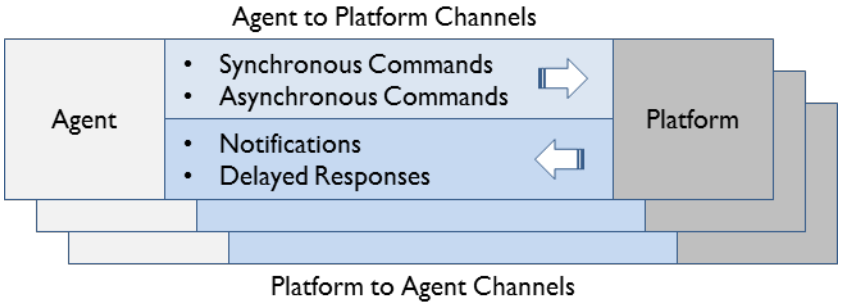

Figure 2: Messages and Channels

Each agent has dedicated channels, which are used to send messages to, and receive messages from, the platform. Each channel is a bidirectional communication pipe between the agent and the platform, except for FastChannels which are unidirectional. For a given channel either the agent or the platform is the requester, or initiator, of communications. The requester can place a message on a channel. At the other end, the completer processes the message, and if the

channel is not a FastChannel, it might place return data on the channel as a response. Depending on which entity is the requester, a channel is one of two types:

- On **A**gent to **P**latform (A2P) channels, the agent is the requester.
- On **P**latform to **A**gent (P2A) channels, the platform is the requester.

Each agent can have one or more A2P channels and one or more P2A channels. A2P channels must be distinct from P2A channels. This allows clear ownership of channels between the agent and the platform. Channels must be dedicated to a specific agent and cannot be shared among other agents. Hence, the maximum number of agents that can co-exist in a system at any given time can be no more than the number of available channels.

The platform considers that all communication over a channel is with a unique agent bearing a fixed agent identifier. This notion enables the platform to identify which agents are communicating with it. The platform statically assigns an agent identifier to every channel. An agent can discover the identifier assigned to it through the channel that the agent owns. This discovery is done using the Base protocol.

The properties of channels are specific to the transport that is used to send messages. An A2P transport might support interrupt-driven communication to send messages, where the platform generates an interrupt when it processes the message. The interrupt alerts the agent that the channel can now be used to send a further message. For a P2A transport, the agent might trigger an interrupt to the platform when it has processed the message sent by the platform. This informs the platform that the channel is now free and can be used to send a further message. Alternatively, a transport might only support polling-based communications. A transport can also support both methods and allow the agent to choose.

Messages are used by agents to make requests to the platform. The messages can carry various parameters, including an identifier for the requested operation. In turn, the platform carries out the requested operation, and might generate data in response to the message. From this point of view, messages are analogous to remote procedure calls, which can carry various parameters, and can also provide return data. The platform can also send messages to an agent, typically to indicate completion of a long job, or to notify of an event.

Messages that are sent by agents on A2P channels are known as commands and fall into two categories:

- **Synchronous**
  The command blocks and the platform responds back when the requested work has completed. It is implementation defined whether or not the transport channel is blocked till the response is received by the agent. The platform responds to these commands over the A2P channel that was used to send them.
- **Asynchronous**
  The platform schedules the requested work to complete later in time. Therefore, these commands return almost immediately to the calling agent, freeing the channel for new commands. The response to an asynchronous command indicates success or failure in the ability to schedule the requested work. When the work has been completed, the platform can send an additional delayed response message to the client over a P2A channel.

Messages that the platform can send to an agent over P2A channels also fall into two categories:

- **Delayed response**
  Messages sent to indicate completion of the work that is associated with an asynchronous command.
- **Notifications**
  These messages provide notifications of events taking place in the platform. Events might include changes in power state, performance state, or other platform status.

FastChannels do not support synchronous commands, delayed responses, or notifications.

3.1.2 Message format

Messages are analogous to remote procedure calls, and therefore must be representative of the operation being requested, and any parameters or return values thereof.

Each message carries a message header, which identifies the operation being requested. Each message belongs to a protocol. Therefore, the header of the message includes an 8-bit protocol identifier. This is known as the protocol_id.

Within a protocol, each message is associated with a unique 8-bit identifier. This is known as the message_id.

A message can take several 32-bit arguments and can provide 32-bit return values. All data values, parameters, message headers, and return arguments are expressed in little endian format. The endianness rule does not apply to strings. For all messages, any reserved field is set to zero.

Values for the protocol_id are described in Table 2.

Table 2: Protocol identifiers

protocol_id	Description
0x0 - 0xF	Reserved
0x10	Base protocol
0x11	Power domain management protocol
0x12	System power management protocol
0x13	Performance domain management protocol
0x14	Clock management protocol
0x15	Sensor management protocol
0x16	Reset domain management protocol
0x17	Voltage domain management protocol
0x18	Power capping and monitoring protocol
0x19	Pin Control protocol
0x1A	MPAM-Fb protocol, see [MPAMFb]
0x1B	System Telemetry protocol
0x1C-0x7F	Reserved for future use by this specification
0x80-0xFF	Reserved for vendor or platform-specific extensions to this interface

For all protocols and all transports using standard channels, messages are sent to the platform using a 32-bit message header, which is described in Table 3. FastChannels do not use a message header as FastChannels are unique to each message.

Table 3: Message header format

Field	Mnemonic	Description
Bits[31:28]	-	Reserved, must be zero.
Bits[27:18]	token	Token.
Bits[17:10]	protocol_id	Protocol identifier.
Bits[9:8]	message_type	Message type. Value of 0x1 is reserved for future use.
Bits[7:0]	message_id	Message identifier. Values 0x10-0x1F are reserved for messages common across all protocols.

Commands

All commands, synchronous or asynchronous, have a message type of 0.

How the token field is used is entirely up to the caller. However, when a command returns, the platform must return the whole message header unmodified. The message header must always be the first parameter that is sent by an agent and returned by the platform.

In addition to the message header, commands return status codes and can return more data. Any command that is sent with an unknown protocol_id or message_id must be responded to with a return value of NOT_SUPPORTED as the status code. FastChannels do not return any status codes since they are unidirectional. Status codes are provided in Section 3.1.4.

Delayed responses

Delayed responses have a message type of 2.

Delayed response messages are sent by the platform to the agent to indicate completion of work that was requested by an asynchronous command. The message header that is associated with a delayed response uses the format that is described in Table 3. The message_id of a delayed response matches that of its associated asynchronous command. The token in the message header matches the token of the associated asynchronous command. The payload that is associated with a delayed response includes a status code, and additional data depending on the command.

Notifications

Notifications have a message type of 3.

Notifications provide a mechanism for the platform to inform agents about events taking place in the platform. Optionally, the implementation can provide information about which agent caused an event. To this end, a notification payload carries an agent identifier, agent_id, as its first parameter. The agent_id is an integer identifier that can be used to codify the agent that generated an event. The agent_id uses the following rules:

- A value of 0 identifies the platform itself.
- Where implemented, agent_ids are sequential and start from one.
- Agent identifiers and their mapping to other components are platform specific. Where necessary, this must be described to operating system through firmware table technologies such as FDT or ACPI.
- If agent identification is not supported, the implementation must set the agent_id to zero in notifications to indicate that the notification has been issued by the platform itself.

The platform is not required to guarantee sending a notification to an agent for every occurrence of the same event. If several events of the same type occur in quick succession, the platform is allowed to only issue a notification for the last occurrence of that event.

3.1.3 Protocol discovery

This specification encompasses various protocols. However, not every protocol has to be present in an implementation, because not every protocol is relevant for every market segment. Furthermore, the platform chooses which protocols it exposes to a given agent. The only protocol that must be implemented is the Base protocol. The Base protocol is used by an agent to discover which protocols are available to it.

All protocols, whether they are generic or vendor specific, must mandatorily implement three special messages with message_ids of `0x0`, `0x1`, and `0x2` respectively, as described in Table 4.

Table 4: Required messages

message_id	Message	Description
`0x0`	PROTOCOL_VERSION	Returns the version of protocol.

message_id	Message	Description
0x1	PROTOCOL_ATTRIBUTES	Returns properties that are associated with the protocol implementation.
0x2	PROTOCOL_MESSAGE_ATTRIBUTES	Takes a message_id as a parameter and returns implementation details specific to that message.

All protocols, including vendor specific, are strongly recommended to implement message_id 0x10 (NEGOTIATE_PROTOCOL_VERSION) intended to negotiate support for agents compliant with older versions of a protocol.

Protocols might implement additional messages.

Protocol versioning uses a 32-bit unsigned integer, where the upper 16 bits are the major revision, and the lower 16 bits are the minor revision.

The following rules apply to the version numbering:

- Higher numbers denote newer versions.
- Different major revision values indicate possibly incompatible messages. For two protocol versions, A and B, which differ in major revision, and where B is higher than A, the following might be true:
  - B can remove messages that were present in A.
  - B can add new messages that were not present A.
  - B can modify the behavior or parameters of messages that are also present in A.
- Minor revisions allow extensions but must retain compatibility. For two protocol versions, A and B, that differ only in the minor revision, and where B is higher than A, the following must hold:
  - Every message in A must also be present in B, and work with compatible effect.
  - It is possible for revision B to have a higher message count than revision A.

3.1.4 SCMI status codes

Messages can return status codes to the sender. Negative 32-bit integers are used to return error status codes. Values 0 to -127 are reserved by this specification. Values below -127 can be used for vendor-specific errors.

Table 5 describes the error codes for SCMI messages.

Table 5: Status codes

Status code	Description
0	SUCCESS
-1	NOT_SUPPORTED
-2	INVALID_PARAMETERS
-3	DENIED
-4	NOT_FOUND

Status code	Description
-5	OUT_OF_RANGE
-6	BUSY
-7	COMMS_ERROR
-8	GENERIC_ERROR
-9	HARDWARE_ERROR
-10	PROTOCOL_ERROR
-11	IN_USE
-12	PARTIAL_ERROR
-13 to -127	Reserved
\< -127	Vendor specific

The specification of each SCMI message describes which error codes are appropriate to that message. However, unless otherwise specified, the following status codes apply to all command messages that are sent from an agent to the platform:

Code	Description
SUCCESS	Successful completion of the command.
NOT_SUPPORTED	The command or feature is not supported or is supported but not within the calling agent's view of the platform.
INVALID_PARAMETERS	One or more parameters passed to the command are invalid or beyond legal limits.
DENIED	The caller is not permitted to perform the specific action, such as accessing a resource or feature that it is not allowed to use.
NOT_FOUND	The entity that is being accessed does not exist. Examples includes non-existent or invalid commands, resources such as power domains, clocks, or sensors.
OUT_OF_RANGE	Requested settings are outside the legal range under the current operating state or condition. NOTE: Legal values can be different for different operating states of the system, hence a setting can be legal at a given point in time, and yet illegal at another. The operating state of the platform can change because of external factors.
BUSY	The platform is out of resources and thus unable to process a command. Arm strongly recommends that the implementation ensures that sufficient resources are available to the platform to handle the more frequently issued commands in order to guarantee availability of service. In particular, the platform must guarantee service for the following commands: • System power protocol commands • AP/domain power management commands. • Reset domain commands An agent receiving this status code must consider the system as non-functional and might attempt recovery through a system restart.

COMMS_ERROR	The message could not be correctly transmitted. Possible causes could include command buffer overflows because of flow control on the message transport. This error is a property of the transport.
GENERIC_ERROR	The command failed to be processed owing to an unspecified fault within the platform.
HARDWARE_ERROR	A hardware error occurred in a platform component during execution of a command.
PROTOCOL_ERROR	Returned when the receiver detects that the caller has violated the protocol specification.
IN_USE	The resource is currently in use by the platform or another agent and cannot be operated upon.
PARTIAL_ERROR	The command was only partially processed by the platform.

3.2 Base protocol

This protocol describes the properties of the implementation and provides generic error management. The Base protocol provides commands to:

- Describe protocol version.
- Discover implementation attributes and vendor identification.
- Discover which protocols are implemented.
- Discover which agents are in the system.
- Register for notifications of platform errors.
- Configure the platform to control and modify an agent's visibility of platform resources and commands.

This protocol is mandatory.

3.2.1 Agent-specific permission configuration and reset

Where the system has multiple agents, the Base protocol provides commands that optionally allow a trusted agent to configure the access permissions of other agents. An agent should not be able to discover resources and commands that it does not have access to. If an agent tries to access resources that it does not have access to, the platform returns a DENIED or NOT_SUPPORTED response.

In a system comprising multiple agents, there is typically one trusted agent which has elevated privileges to configure and control the access rights of other agents in the system. Nominating a trusted agent is an implementation defined choice that must consider the deployment use case. Arm recommends that only trusted agents have access to the Base Protocol commands to configure agent specific permissions. A trusted agent may be in the Root, Secure or Non-secure Security state. Non-trusted agents should not be able to modify access permissions of other agents.

Platform resources can be assigned to Root, Secure or Non-secure. Only a trusted agent in the Root or Secure Security state should be able to modify access permissions of Secure platform resources. Similarly, only a trusted agent in the Root Security state should be able to modify access permissions of Root platform resources. A trusted agent in the Non-secure Security state cannot modify the access permissions of Root or Secure platform resources.

The system boots with default permission configurations for each agent. Typically, this might ensure that agents in the Non-secure Security state do not have access to Root or Secure platform resources. Thereafter, the trusted agent might set up additional access permissions.

In a virtualized system, a Virtual Machine (VM) is an example of a non-trusted agent in the Non-secure Security state, and the hypervisor is an example of a trusted agent in the Non-secure Security state. Using agent-specific permission configuration, the hypervisor can set up fine grained Non-secure resource access permissions for Virtual Machines. The hypervisor discovers the agent identifier of the channel that it wants to assign to a VM. The hypervisor then sets up access permissions of the agent identifier associated with that channel and assigns the channel to the VM. The VM can now discover the agent identifier, and only access those protocols and platform resources which have been permitted by the hypervisor.

3.2.1.1 Device specific access control

A platform usually includes a set of devices, or peripherals, for example Graphics Processing Units (GPUs), UART, or USB. If a platform has multiple agents, all agents might not have access to all the devices in the platform. The base protocol provides the BASE_SET_DEVICE_PERMISSIONS command to configure the devices that an agent has access to. A device, in this context, might also be a logical aggregation of platform components. The definition of a device is the responsibility of the platform firmware and depends on the use case and the platform itself.

The platform must track all the resources that constitute a device. Platform resources refer to power domains, performance domains, clocks, sensors, reset domains and voltage domains. A device might span multiple domains. Also, multiple devices might share the same domain. An agent should be able to access resources associated with a device, only when the agent has permissions to access the device itself. When a resource is shared among multiple devices, the resource must be maintained in a state that fulfils the requirements of all the devices that share it.

3.2.1.2 Protocol specific access control

The Base protocol allows a trusted agent to restrict the protocols that non-trusted agents can use to access the platform resources that are associated with a specific device.

This restriction is achieved by the BASE_SET_PROTOCOL_PERMISSIONS command. The platform should generate a DENIED or NOT_SUPPORTED response if an agent tries to use a restricted protocol to access the platform resources that are associated with the specific device.

3.2.1.3 Agent specific configuration reset

The Base protocol provides the BASE_RESET_AGENT_CONFIGURATION command to reset all the platform resource configurations that are requested by an agent and tracked by the platform. This command can also be used to reset agent-specific permission configurations to access devices and protocols.

A trusted agent might be allowed to reset the configuration of any agent in the system. However, a trusted agent in Non-secure Security state should not be allowed to reset Secure platform resource permissions or configurations. Non-trusted agents should not be allowed to reset the configuration of other agents. A non-trusted agent might only request configuration reset for itself.

Agent configuration reset should not be confused with system reset which is achieved through System power management protocol, or the reset of a domain which is achieved through the Reset domain management protocol. Agent specific configuration reset might typically be used in a scenario in which the trusted agent might want to remove an agent's access to all devices previously assigned to it. Agent specific configuration reset might also be useful when an agent has become unresponsive, and the trusted agent needs to tell the platform to clean up that agent's resource configurations.

3.2.2 Commands

3.2.2.1 PROTOCOL_VERSION

This command returns the version of this protocol. For this version of the specification, the value that is returned must be `0x20001`, which corresponds to version 2.1.

message_id: `0x0`
protocol_id: `0x10`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this value must be `0x20001`.

3.2.2.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: 0x10
protocol_id: 0x10
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.2.2.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details that are associated with this protocol.

message_id: 0x1
protocol_id: 0x10
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 attributes	**Bits[31:16]** Reserved, must be zero. **Bits[15:8]** Number of agents in the system. **Bits[7:0]** Number of protocols that are implemented, excluding the Base protocol.

If the platform does not support agent discovery, then it reports the number of agents in the system as zero, and all notifications carry a zero in the agent_id field.

3.2.2.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

message_id: 0x2
protocol_id: 0x10
This command is mandatory.

Parameters

Name	Description

uint32 message_id	message_id of the message.

Return values

Name	Description
int32 status	One of the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is not provided by this platform implementation. See Section 3.1.4 for status code definitions.
uint32 attributes	Flags that are associated with a specific command in the protocol. For all commands in this protocol, this parameter has a value of 0.

3.2.2.5 BASE_DISCOVER_VENDOR

This command provides a vendor identifier ASCII string.

message_id: `0x3`
protocol_id: `0x10`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint8 vendor_identifier [16]	Null terminated ASCII string of up to 16 bytes with a vendor name.

3.2.2.6 BASE_DISCOVER_SUB_VENDOR

On success, this optional command provides a sub vendor identifier ASCII string.

message_id: `0x4`
protocol_id: `0x10`
This command is optional.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint8 vendor_identifier [16]	Null terminated ASCII string of up to 16 bytes with a vendor name.

3.2.2.7 BASE_DISCOVER_IMPLEMENTATION_VERSION

This command provides a vendor-specific 32-bit implementation version. The format of the version number is vendor-specific, but version numbers must be strictly increasing so that a higher number indicates a more recent implementation.

message_id: 0x5
protocol_id: 0x10
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 implementation_version	Format is vendor specific.

3.2.2.8 BASE_DISCOVER_LIST_PROTOCOLS

This command allows the agent to discover which protocols it is allowed to access. The protocol list returned by this call should be in numeric ascending order.

message_id: 0x6
protocol_id: 0x10
This command is mandatory.

Parameters

Name	Description
uint32 skip	Number of protocols to skip.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if a valid list of protocols is found. • INVALID_PARAMETERS: if skip field is invalid. See Section 3.1.4 for more status code definitions.
uint32 num_protocols	Number of protocols that are returned by this call.
uint32 protocols[1+(num_protocols-1)/4]	Array of protocol identifiers that are implemented, excluding the Base protocol, with four protocol identifiers packed into each array element. The PROTOCOL_ATTRIBUTES command can be used to determine the number of protocols implemented.

The following pseudocode illustrates how this command can be used.

```
int status = 0;
int skip = 0;
int total_protocols = 0;
int num_protocols = 0;

uint32 attributes = 0;
uint32* protocols = NULL;
invoke_PROTOCOL_ATTRIBUTES(&status,&attributes);

if (status)
     goto clean_up_and_return;

total_protocols = (attributes & NUM_PROTOCOLS_MASK) >>
                        NUM_PROTOCOLS_SHIFT;
```

```
if (!total_protocols)
     goto clean_up_and_return;

do {
    invoke_BASE_DISCOVER_LIST_PROTOCOLS(skip,
        &status, &num_protocols, protocols);

    if (status)
        goto clean_up_and_return;

    for (int ix = 0; ix < num_protocols; ix++)
    {
        uint8 prot = protocols[ix/4] >> ((ix % 4) * 8);
        add_to_protocol_database(prot);
        skip++;
    }
} while (skip < total_protocols);
```

3.2.2.9 BASE_DISCOVER_AGENT

This optional command allows the caller to discover the name of an agent, described through an ASCII string of up to 16 bytes. A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command.

If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS (0).

Agent identifiers, agent_id, describe agents in the system that can use the SCMI protocols. Not every agent can use all protocols, and some protocols can offer different views to different agents. An agent_id of 0 is reserved to identify the platform itself. If the command is not implemented, the caller does not interpret agent identifiers in notifications, and the platform sets agent_id to zero in notifications. Where supported, agent_id values are sequential, start from one, and are limited by the number of agents that is reported through PROTOCOL_ATTRIBUTES.

If called with an agent_id of 0, the string returned in the name parameter must start with “platform”.

An agent can discover its own agent_id and name by passing agent_id of `0xFFFFFFFF`. In this case, the command returns the agent_id and name of the calling agent.

message_id: `0x7`
protocol_id: `0x10`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent whose identification is requested.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: If a valid agent identifier is found. • NOT_FOUND: if agent_id does not point to a valid agent. See Section 3.1.4 for more status code definitions.

uint32 agent_id	ID of the agent whose identity is requested. This field is: • populated with the agent_id of the calling agent, when the agent_id parameter passed via the command is `0xFFFFFFFF`. • identical to the agent_id field passed via the calling parameters, in all other cases.
uint8 name[16]	Null terminated ASCII string of up to 16 bytes in length.

3.2.2.10 BASE_NOTIFY_ERRORS

An implementation can optionally provide notifications of errors in the platform to an agent that has registered through this command. A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command.

If the command is implemented, then the PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

Error notification is used to notify agents of commands that could not proceed due to unpredictable circumstances, such as internal hardware errors. Further information on the error notification and associated payload is provided in Section 3.2.3.1, which describes the BASE_ERROR_EVENT notification.

message_id: `0x8`
protocol_id: `0x10`
This command is optional.

Parameters

Name	Description
uint32 notify_enable	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Notify enable: • If this value is 0, the platform does not send any BASE_ERROR_EVENT messages to the calling agent. • If this value is 1, the platform sends BASE_ERROR_EVENT messages to the calling agent when an error is detected. For details on BASE_ERROR_EVENT notification, see Section 3.2.3.1.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • INVALID_PARAMETERS: if notify_enable contains illegal or incorrect values. See Section 3.1.4 for more status code definitions.

3.2.2.11 BASE_SET_DEVICE_PERMISSIONS

This command is used to indicate to the platform whether an agent has permissions to access devices, as specified by a device identifier. An agent can only operate on devices to which it has access, and by extension can only operate on the power, performance, clock, sensor, reset and voltage domains that are associated with that device. At system boot, the default device-specific access permission of an agent is IMPLEMENTATION defined. Arm recommends that only trusted agents in the system are given permission to invoke this command.

The Base protocol does not cover the discovery of device identifiers for devices in a platform. This information is provided to the caller by way of firmware tables in FDT or ACPI.

A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command. If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

message_id: `0x9`
protocol_id: `0x10`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the Agent.
uint32 device_id	Identifier of the device.
uint32 flags	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Access Type: This bit defines the permissions of the agent to access platform resources associated with the device. • If set to 0, deny agent access to the device. • If set to 1, allow agent access to the device.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the device permissions was set successfully for the agent specified by agent_id. • NOT_FOUND: if agent_id or device_id does not exist. • INVALID_PARAMETERS: if flags field is invalid. • NOT_SUPPORTED: if the command is not supported. • DENIED: if the calling agent is not allowed to set the permissions of the agent specified by agent_id. See Section 3.1.4 for more status code definitions.

3.2.2.12 BASE_SET_PROTOCOL_PERMISSIONS

An agent can have access to multiple devices. The agent uses commands to access platform resources that are associated with those devices. The command BASE_SET_PROTOCOL_PERMISSIONS is used to indicate to the platform whether an agent has permissions to use a protocol to access the platform resources that are associated with a specific device. This command cannot be used to change the agent's permissions to access a device. This command only affects the protocols which the agent can use to access the platform resources that are associated with a particular device. At system boot, the default per-device protocol specific access permissions of an agent are IMPLEMENTATION defined.

Arm recommends that only trusted agents in the system are given permissions to invoke this command.

A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command. If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

message_id: `0xA`
protocol_id: `0x10`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the Agent.
uint32 device_id	Identifier of the device.
uint32 command_id	**Bits[31:8]** Reserved, must be zero. **Bits[7:0]** Protocol ID. • This field should not be set to `0x10`, since it is mandatory to implement the Base protocol for all agents.
uint32 flags	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Access Type. This bit defines the permissions of the agent to use the protocol specified by command_id, to access platform resources that are a part of the device specified by device_id. • If set to 0, deny agent access to the protocol. • If set to 1, allow agent access to the protocol.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the command permissions were set successfully. • NOT_FOUND: if any of agent_id, device_id or protocol_id does not exist. • INVALID_PARAMETERS: if flags field is invalid. • NOT_SUPPORTED: if the command is not supported. • DENIED: if the calling agent is not allowed to set the protocol permissions for the agent specified by agent_id. See Section 3.1.4 for more status code definitions.

3.2.2.13 BASE_RESET_AGENT_CONFIGURATION

This command is used to reset platform resource settings that were previously configured by an agent. Platform resource settings refer to power domain, performance domain, clock, sensors and other settings associated with a device that the agent has access to. This command can also be used to reset agent-specific permission configurations to access devices and protocols.

When this command is received, the platform might need to flush all pending requests from the agent that is undergoing configuration reset. It might also need to wait for requests that are being processed on behalf of the agent to complete. Alternatively, the platform can choose to abort all agent-related transactions in flight and reset its configuration. The platform needs to revert the platform resources that are solely dedicated to the agent into their default state. Shared platform resources need to be moved into a state that continues to meet the requirements of the remaining agents using that resource. Shared platform resources are those which are shared among and used by multiple agents. Agent configuration reset should not be confused with the reset of the platform or its components.

If the Permissions Reset flag is set, the platform resets all the device and protocol access permissions that are configured for the agent. When permission reset completes, IMPLEMENTATION defined platform-specific default permissions are restored for that agent.

Arm recommends that only trusted agents in the system are given permissions to invoke this command for other agents. An agent can invoke this command for itself.

A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command. If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

message_id: `0xB`
protocol_id: `0x10`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the Agent.
uint32 flags	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Permissions Reset: • If set to 0, maintain all access permission settings of the agent. • If set to 1, reset all access permission settings of the agent. This command must always reset the platform resource settings configured by the agent specified by agent_id. Platform resource settings refer to Device, Power Domain, Performance Domain, Clocks, Sensors and other settings configured by the agent specified by agent_id.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the command is processed successfully. • NOT_FOUND: if the agent specified by agent_id does not exist. • INVALID_PARAMETERS: if the flags field is invalid. • NOT_SUPPORTED: if the command is not supported. • DENIED: if the calling agent is not allowed to reset the agent specified by agent_id. See Section 3.1.4 for more status code definitions.

3.2.3 Notifications

3.2.3.1 BASE_ERROR_EVENT

These notifications are sent to any agent that has registered to receive them, provided the platform implements base error notifications.

Errors that are reported by the platform are one of two types:

- **Fatal error**
  Indicates that the platform is no longer able to process commands. The error might be accompanied by the list of messages that were being processed when the failure took place.

- **Non-fatal error**
  Indicates that the platform was not able to process some commands, but it is still operational. The error notification is accompanied by the list of commands that could not be processed.

Fatal error notifications cannot be guaranteed, and the platform must not rely on these notifications as a mechanism to enable recovery.

Error notifications must not be used as mechanism to report that a command cannot be executed as requested due to constraints that arise in normal operation.

On initial boot of an agent, these notifications must be disabled by default to that agent.

message_id: `0x0`
protocol_id: `0x10`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent that caused this notification.
uint32 error_status	**Bit[31]** Fatal. • Set if error is fatal and platform cannot continue. • Cleared if error is non-fatal but commands have failed. **Bits[30:10]** Reserved, must be zero. **Bits[9:0]** Command count, number of commands in the command list. A value of zero is possible if the error cannot be attributed.
{uint32 message_header unit32 status}[N]	Each entry in the command list is a tuple, where the first entry is the message header of the command, and second is an error status code that is associated with the command. The size of the list is specified by the command count sub-field.

3.3 Power domain management protocol

This protocol is intended for management of power states of power domains.

The power domain management protocol provides commands to:

- Describe the protocol version.
- Discover implementation attributes.
- Set the power state of a domain.
- Get the current power state of a domain.
- Optionally get notifications when power domains change state or when an agent requests for a power domain state change.
- Optionally return statistics on residency and usage count of a given power state.

3.3.1 Power domain management protocol background

In this document, a power domain is defined as a group of components that are powered together. For example, a set of components that share a power source, and can only be turned ON or OFF as a group, form a power domain. Power domains have the following properties:

- They can include one or more devices.
- They must at least support the ON and OFF states but can support additional power states.
- In the ON state, the domain is operational and devices within it can run.
- In the OFF state, the domain has no power supplied to it. Devices within it cannot run and lose all context.

Domains can have dependencies on other domains. For example, a parent domain can include a child domain. In such a case, if the child domain is ON, the parent domain is also necessarily ON. Dependencies can also be implicit. For example, a power domain (named A), might be indirectly used by multiple agents, through other power domains which implicitly depend on the power domain A. Such an implicitly used power domain must be in a power state that can ensure service guarantees to all agents.

The protocol does not cover discovery of power states that are supported by a domain, or description of the properties of the states, for example associated latencies, context loss, or domain dependencies. This information is expected to be provided to the caller by way of firmware tables in FDT or ACPI.

Protocol commands take integer identifiers to identify the power domain that they apply to. The identifiers are sequential and start from 0.

The protocol can be used to manage the power state of application processors and devices in the system.

—— Note ——————————

Operating systems that are running on application processors must not directly use SCMI to manage the power state of these processors. Instead, power states for domains that include APs must be managed using PSCI calls from the operating system. When the OSPM calls a PSCI function, the PSCI implementation, which is described in [PSCI, ARMTF], can communicate with the platform using this protocol over Secure or Root channels. This protocol allows SCMI to provide an implementation for PSCI functions designed to manage the power of application processors, such as CPU_DEFAULT_SUSPEND, CPU_SUSPEND, CPU_FREEZE, CPU_ON and CPU_OFF. These functions map to various use cases including idle, secondary core boot, and hot plug. The list does not include system power state transitions such as system shutdown or reset, which are covered by the system power management protocol instead, as described in Section 3.4.

————————————————

Agents that are not running on application processors can register to receive notifications of power state changes to these power domains.

Non-secure channels can be used to manage power domains for devices that do not include application processors, and which are not used by Secure or Root entities in the system. Any agent can register for power state change notifications for these domains.

An implementation can include devices that are intended for use only by Secure entities in the system such as a trusted OS. Power domains for such devices must be managed through Secure channels.

Agents other than the OSPM can manage power domains. In a multi-agent system with multiple domains, several scenarios are possible:

- A power domain is exclusive to an agent.
- A power domain can be shared by multiple agents.

Agents can coordinate with the platform to access power domains, and to perform power management of the domains which it has access to. For AP power domains, the coordination models are analogous to those described in [ACPI] and [PSCI]. For all combinations of power domains and agents, platform policy dictates which agents can access which power domains, and whether a power domain is shared or exclusive. This policy is guided by the required agent functionality while maintaining the security, confidentiality, and integrity of other agents in the system. When a power domain is shared among multiple agents, the power domain must be in a power state that can ensure service guarantees to all agents. This is typically the shallowest power state requested by any of the agents using the power domain

For all messages in this protocol, the interpretation of the power state parameter is specific to the combination of the agent and the power domain that it is managing. A power domain with Application Processors that is managed by a PSCI agent must support representation of the power state parameter based on definitions in [PSCI]. On the other hand, for power domains pertaining to devices, the power state parameter must minimally represent two pre-defined states, ON and OFF. Power state encoding for device power domains is described in Table 6.

Table 6: Power State Parameter Layout for Device Power Domains

Bit field	Description
31	Reserved. Must be zero.
30	StateType If set to 0, indicates that context is preserved. If set to 1, indicates that context is lost.
29:28	Reserved. Must be zero.
27:0	StateID A value of zero when StateType is set to 0 represents the ON state. A value of zero when StateType is set to 1 represents the OFF state. All other values are IMPLEMENTATION_DEFINED.

3.3.2 Commands

3.3.2.1 PROTOCOL_VERSION

On success, this command returns the Protocol version. For this version of the specification, the return value must be `0x30001`, which corresponds to version 3.1.

message_id: `0x0`
protocol_id: `0x11`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, the value must be `0x30001`.

3.3.2.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: `0x10`
protocol_id: `0x11`
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.3.2.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: `0x1`
protocol_id: `0x11`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 attributes	**Bits[31:16]** Reserved, must be zero. **Bits[15:0]** Number of power domains.

uint32 statistics_address_low	The lower 32 bits of the physical address where the statistics shared memory region is located. This value should be 64-bit aligned. The address must be in the memory map of the calling agent. This field is invalid and must be ignored if the statistics_len field is set to 0. The statistics shared memory region is described in Section 3.3.4.
uint32 statistics_address_high	The upper 32 bits of the physical address where the statistics shared memory region is located. The address must be in the memory map of the calling agent. This field is invalid and must be ignored if the statistics_len field is set to 0. The statistics shared memory region is described in Section 3.3.4.
uint32 statistics_len	The length in bytes of the statistics shared memory region. A value of 0 in this field indicates that the platform doesn't support the statistics shared memory region.

Note: Statistics capability has been marked for deprecation and is being replaced by system telemetry.

3.3.2.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

This command can be used to inquire if power state change notifications are supported, by passing POWER_STATE_NOTIFY or POWER_STATE_CHANGE_REQUESTED_NOTIFY message identifier to the call.

If the platform returns SUCCESS, then it supports power state change notifications. Otherwise, if the platform returns NOT_FOUND, then it is an indication that notifications are not implemented, or that notifications are not available to the calling agent. The notifications commands are described in Section 3.3.2.8 and Section 3.3.3.1.

message_id: `0x2`
protocol_id: `0x11`
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is invalid or not implemented. See Section 3.1.4 for more status code definitions.
uint32 attributes	Flags that are associated with a specific command in the protocol. In the current version of the specification, this value is always 0.

3.3.2.5 POWER_DOMAIN_ATTRIBUTES

This command returns the attribute flags associated with a specific power domain.

message_id: `0x3`
protocol_id: `0x11`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the domain. Domain identifiers are limited to 16 bits, and the upper 16 bits of this field are ignored by the platform.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid power domain attributes are returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.
uint32 attributes	**Bit[31]** Power state change notifications support. • Set to 1 if power state change notifications are supported on this domain. • Set to 0 if power state change notifications are not supported on this domain. **Bit[30]** Power state asynchronous support. • Set to 1 if power state can be set asynchronously. • Set to 0 if power state cannot be set asynchronously. **Bit[29]** Power state synchronous support. • Set to 1 if power state can be set synchronously. • Set to 0 if power state cannot be set synchronously. **Bit[28]** Power state change requested notifications support. • Set to 1 if power state change requested notifications are supported on this domain. • Set to 0 if power state change requested notifications are not supported on this domain. **Bit[27]** Extended power domain name. • If set to 1, the power domain name is greater than 16 bytes. The extended power domain name is provided by POWER_DOMAIN_NAME_GET command which is described in Section 3.3.2.10. • If set to 0, extended power domain name is not supported. **Bits[26:0]** Reserved, must be zero.
uint8 name[16]	Null-terminated ASCII string of up to 16 bytes in length describing the power domain name. When Bit[27] of attributes field is set to 1, this field returns the NULL terminated lower 15 bytes of the power domain name.

For some agents, the platform might only allow registration and receipt of notifications for power domains and disallow setting of power states of those domains.

3.3.2.6 POWER_STATE_SET

This command allows an agent to set the power state of a power domain. Power domains can be managed synchronously or asynchronously:

- **Synchronous Mode**
  A call with valid parameters completes and frees the channel when the domain has transitioned to the desired power state.
- **Asynchronous Mode**
  The call completes immediately, and the caller can register for notifications if it wishes to observe the power state transition. These notifications are described in Section 3.3.3.1.

When this command is used for power domains that include application processors, the Async flag is ignored. This call must return to the calling AP before that AP is powered down. Following this call, the AP executes some instructions before invoking a *Wait for Interrupt* (WFI) instruction [ARM]. The platform controller that implements SCMI begins the transition to the required power state when it observes the WFI. The method used by the platform controller to observe the WFI is IMPLEMENTATION DEFINED. For these power domains, this protocol can be used to implement PSCI CPU_SUSPEND, CPU_ON, CPU_FREEZE, CPU_DEFAULT_SUSPEND and CPU_OFF functions.

A power domain can contain other power domains. If the caller wants to change the state of a power domain and one of its parents, the power domain parameter must identify the child. The required power state for the child domain, and its parents, must be encoded in the power state parameter. How this is encoded in the power_state parameter is IMPLEMENTATION DEFINED.

message_id: `0x4`
protocol_id: `0x11`
This command is mandatory.

Parameters

Name	Description
uint32 flags	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Async flag. • Set to 1 if power transition must be done asynchronously. • Set to 0 if power state transition must be done synchronously. The async flag is ignored for application processor domains.
uint32 domain_id	Identifier for the power domain.
uint32 power_state	Platform-specific parameter identifying the power state of the domain. For device power domains, this parameter is encoded as described in Table 6.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: for a power domain that can only be set synchronously, this status is returned after the power domain has transitioned to the desired state. For a power domain that is managed asynchronously, this status is returned if the command parameters are valid, and the power state change has been scheduled. • NOT_FOUND: if the power domain identified by domain_id does not exist. • INVALID_PARAMETERS: if the requested power state does not represent a valid state for the power domain that is identified by domain_id. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to set the state of this power domain. An example would be if this power domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.

3.3.2.7 POWER_STATE_GET

This command allows the calling agent to request the current power state of a power domain.

Note

It is possible for the power_state value returned by this command to be stale by the time the command completes, as another state change request could have been initiated and completed in the interim.

message_id: `0x5`
protocol_id: `0x11`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the power domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the current power state is returned successfully. • NOT_FOUND: if domain_id does not point to a valid power domain. Ses Section 3.1.4 for more status code definitions.
uint32 power_state	Platform-specific parameter identifying the power state of this domain. For device power domains, this parameter is encoded as described in Table 6.

3.3.2.8 POWER_STATE_NOTIFY

This command allows the caller to request notifications from the platform for state changes in a specific power domain. These notifications are sent using the POWER_STATE_CHANGED notification, which is described in Section 3.3.3.1.

Notification support is optional, and PROTOCOL_MESSAGE_ATTRIBUTES must be used to discover whether this command is implemented.

These notifications must be disabled by default during initial boot of the platform.

message_id: `0x6`
protocol_id: `0x11`
This command is optional.

Parameters

Name	Description
uint32 domain_id	Identifier for the power domain.
uint32 notify_enable	**Bits[31:1]** Reserved must be zero. **Bit[0]** Notify enable. This bit can have one of the following values: • 0, which indicates that the platform does not send any POWER_STATE_CHANGED messages to the calling agent. • 1, which indicates that the platform does send POWER_STATE_CHANGED messages to the calling agent when a domain changes power state. See Section 3.3.3.1 for POWER_STATE_CHANGED notification.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if domain_id does not point to a valid domain. • INVALID_PARAMETERS: if notify_enable specifies values that are either illegal or incorrect. See Section 3.1.4 for more status code definitions.

3.3.2.9 POWER_STATE_CHANGE_REQUESTED_NOTIFY

This command allows the caller to receive notifications from the platform, when the platform receives a request from another agent to change the state of a power domain.

These notifications are sent using POWER_STATE_CHANGE_REQUESTED notification described in Section 3.3.3.2.

POWER_STATE_CHANGE_REQUESTED notifications are useful for the co-operative management of power domains that are shared among agents. When a power domain is shared among agents, the platform maintains the power domain in a state that meets the requirements of all the agents that are sharing it.

For example, the POWER_STATE_CHANGE_REQUESTED notification can be used when a request is made by one agent to turn off a shared power domain. The platform will not act on this request if other agents have requested the same power domain to be active. The platform will notify the other agents sharing the power domain through the POWER_STATE_CHANGE_REQUESTED notification, if the other agents have subscribed to it. This notification enables the other agents to allow the power state transition of the shared power domain, by voluntarily relinquishing the use of the shared power domain. The decision to voluntarily relinquish the use of a shared power domain is based on an implementation-defined policy.

Notification support is optional, and PROTOCOL_MESSAGE_ATTRIBUTES must be used to discover whether this command is implemented.

These notifications must be disabled by default during initial boot of the platform.

message_id: 0x7
protocol_id: 0x11
This command is optional.

Parameters

Name	Description
uint32 domain_id	Identifier for the power domain.
uint32 notify_enable	**Bits[31:1]** Reserved must be zero. **Bit[0]** Notify enable. This bit can have one of the following values: • 0, which indicates that the platform does not send POWER_STATE_CHANGE_REQUESTED messages to the calling agent. • 1, which indicates that the platform sends POWER_STATE_CHANGE_REQUESTED messages to the calling agent when another agent requests for a change in the state of the power domain. See Section 3.3.3.2 for the details on POWER_STATE_CHANGE_REQUESTED notification.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if domain_id does not point to a valid domain. See Section 3.1.4 for more status code definitions.

3.3.2.10 POWER_DOMAIN_NAME_GET

This command returns the extended name associated with a specific power domain.

message_id: 0x8
protocol_id: 0x11
This command is mandatory only if extended power domain name is supported.
Extended power domain name is supported when Bit[27] of attributes field of POWER_DOMAIN_ATTRIBUTES command is set to 1. Refer to Section 3.3.2.5 for more details.

Parameters

Name	Description
uint32 domain_id	Identifier for the power domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid power domain extended name is returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.

uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint8 ext_name[64]	Null-terminated ASCII string of up to 64 bytes in length describing the power domain extended name.

3.3.3 Notifications

3.3.3.1 POWER_STATE_CHANGED

If an agent has registered to receive power state change notifications for the power domain that is identified by domain_id, the platform sends these notifications to that agent when the power domain state changes, including transitions to an ON state.

Note that notified power states might not match those requested by the agent that is notified. The power state that is finally selected by the platform might differ from that requested by an agent, due to coordination with other requests on the same domain.

message_id: `0x0`
protocol_id: `0x11`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent that caused the power transition.
uint32 domain_id	Identifier of the power domain whose power state was changed.
uint32 power_state	The power state that the power domain transitioned to. These notifications take place when the transition has completed.

3.3.3.2 POWER_STATE_CHANGE_REQUESTED

An agent might have registered, via POWER_STATE_CHANGE_REQUESTED_NOTIFY, to receive notifications when the platform receives a request from a different agent to change the power state of a power domain. The platform sends this notification to the interested agent when such a request is received by it.

For more details on how POWER_STATE_CHANGE_REQUESTED notifications can be used, see Section 3.3.2.9.

message_id: `0x1`
protocol_id: `0x11`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent that requested the power transition.
uint32 domain_id	Identifier of the power domain whose power state change is being requested.
uint32 power_state	The power state requested.

3.3.4 Power state statistics shared memory region

Note: Statistics capability has been marked for deprecation and is being replaced by system telemetry.

Optionally, the platform can provide a statistics shared memory region that is associated with the power state protocol. Whether support is present is indicated by the PROTOCOL_ATTRIBUTES command, which is described in Section 3.3.2.3. The PROTOCOL_ATTRIBUTES command also provides the address and the size of the shared memory region. The region provides usage counts and residency information for each power state that is used by each power state domain. The memory must be accessible from the Non-secure world, and OSPM must map it as non-cached normal memory or device memory. For a given power domain, and for each power state in a domain, statistics in the shared memory region track the number of times the state has been used and the amount of time the domain has been in the state. The statistics must be updated regardless of the agent in the system that placed a domain into a given power state. After a system reset or shutdown, all the statistics must be initialized to zero. Time measurements are in microseconds.

The design of the statistics shared memory region allows the platform implementation to choose which power domains are included. However, if a domain is included, all its power states must be represented, including time that is spent in an ON state.

The format of the statistics shared memory region is described in Table 7.

Table 7: Power state statistics shared memory region

Field	Byte Length	Byte Offset	Description
Signature	0x4	0x0	0x504F5752 ('POWR').
Revision	0x2	0x4	For this revision, this field must be 0x1.
Attributes	0x2	0x6	For this revision, this field must be zero.
Number of domains	0x2	0x8	Number of domains for which statistics are collected.
Reserved	0x2	0xA	Must be zero.
Match Sequence	0x4	0xC	The match sequence populated by the platform as described in Section 3.3.4.1.
Power domain offset array	0x4 × (Total number of power domains)	0x10	For each power domain, this array provides a 4-byte offset, from the start of the shared memory area, to the memory location of the power domain entry in the data section. The entry is described in Table 8. A value of zero for the offset of a given power domain indicates that statistics are not collected for that domain.
Power domain data section	--	--	This area must start at an offset of 0x10 + 0x4 × (Number of power domains), or higher.

3.3.4.1 Match Sequence

Race conditions can arise between write accesses by the platform and read accesses by the agent during a statistics shared memory region update. This can lead to stale or invalid values being read by the agent. The match sequence is used to detect such race conditions.

The match sequence is a 4 byte non-zero unsigned value that is populated by the platform and is read by the agent. The match sequence should be an even number at boot. The sequence should be incremented by one, once at the start of, and once at the end of a statistics shared memory region update. This ensures that the match sequence is an odd number when the statistics shared memory region is being updated and is an even number otherwise. Care should be

taken to handle the case when the match sequence wraps to zero. In such a scenario, the platform can choose to reset the match sequence to its boot time value.

A flow chart depicting the steps involved when the platform updates the statistics shared memory region is shown in Figure 3. The match sequence is an even number when the statistics shared memory region is not being updated. The platform increments the match sequence by one at the beginning of a statistics shared memory region update process. As a result, the match sequence becomes an odd number when the statistics shared memory region is being updated. The platform then updates the statistics shared memory region. When the platform has completed updating the statistics shared memory region, it again increments the match sequence by one. As a result, the match sequence becomes an even number when the update process is complete. The match sequence before and after the update differ from each other. An odd value of the match sequence along with a mismatch of values can be used by an agent to detect invalid reads from the statistics shared memory region.

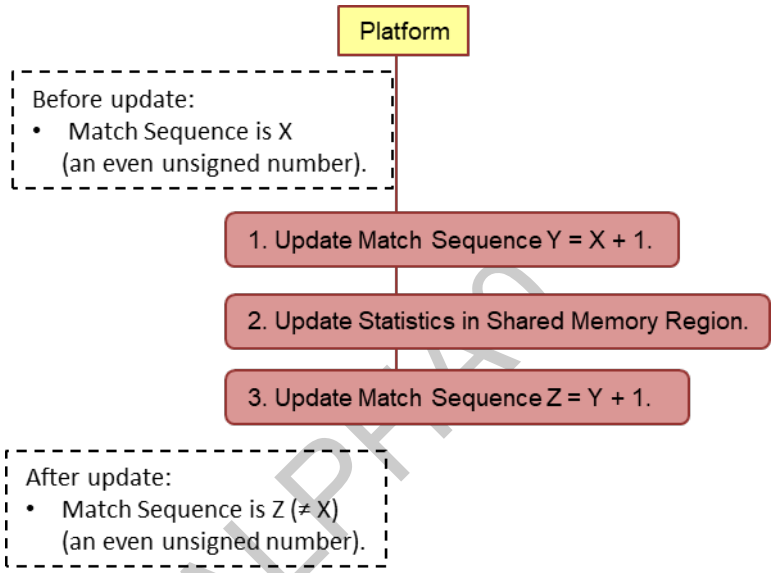

Figure 3: Statistics Shared Memory Region Update by Platform

A flow chart depicting the steps involved when an agent reads the statistics shared memory region is shown in Figure 4. The agent reads the match sequence before reading the statistics shared memory region. If the value read is odd, the agent aborts the read and retries till the match sequence read is an even number. The agent reads the match sequence again at the end of reading the statistics shared memory region. If the match sequences before and after the statistics shared memory region read are identical, the statistic was read successfully. Otherwise further reads must be done till the start match sequence and the end match sequence are equal and they are even numbers.

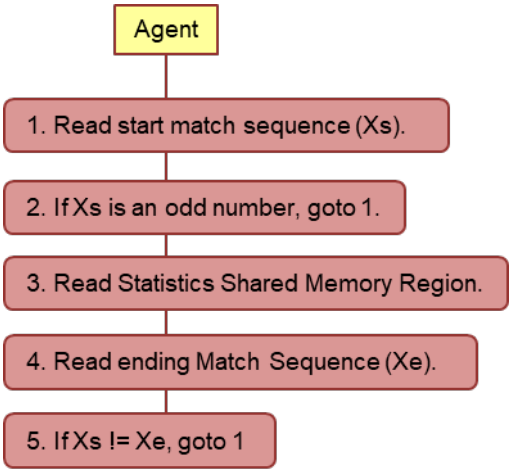

Figure 4: Statistics Shared Memory Region Read by Agent

3.3.4.2 Power domain data section

The power domain data section contains an entry for each power domain for which statistics are collected. The format for each entry is described in Table 8.

Table 8: Power domain entry

Field	Byte Length	Byte Offset	Description
Number of power states	0x2	0x0	Number of power state entries in the power state array.
Current power state Index	0x2	0x2	Index into power state array for current power state.
Reserved	0x4	0x4	Must be zero.
Time of last change	0x8	0x8	Timestamp in microseconds, from the beginning of the current boot, of the last power state transition, including to a running state.
Power state array	N × 0x18	0x10	Where N is the number of power states. Described in Table 9.

The format for each entry in the power state array is described in Table 9.

Table 9: Power state entry

Field	Byte Length	Byte Offset	Description
Power state	0x4	0x0	Identifier for the power state.
Reserved	0x4	0x4	Must be zero.

Field	Byte Length	Byte Offset	Description
Usage count	0x8	0x8	Number of times this domain has entered the power state. This value must be updated when the domain transitions into the power state.
Residency	0x8	0x10	Amount of time in microseconds domain has been resident in the power state. This value must be updated when the domain transitions out of the power state.

3.4 System power management protocol

This protocol is intended for system shutdown, suspend, and reset.

The system power protocol provides commands to:

- Describe the protocol version.
- Discover implementation attributes.
- Shut down the system.
- Suspend the system.
- Reset the system.
- Request a graceful shutdown or reset.
- Allow an agent to forcibly power down or reset the system.

3.4.1 System power management protocol background

The OSPM must be able to power down or reset the whole system it is running on. ACPI provides S states (S1-S5) for this purpose. In turn, PSCI provides SYSTEM_RESET, SYSTEM_RESET2, SYSTEM_SUSPEND and SYSTEM_OFF. On some systems, other agents might be required to initiate a system power down or reset. This protocol is designed to allow more than one agent to request these types of system power transitions. It is envisaged that, in the common case, there might be three agents:

- On application processors, a PSCI implementation. The PSCI implementation fulfills OSPM calls to SYSTEM_OFF, SYSTEM_SUSPEND, SYSTEM_RESET and SYSTEM_RESET2 functions. To do so, the PSCI implementation uses the SCMI protocol to request system power down or reset transitions.
- The management agent or privileged agent:
  - Particularly in enterprise systems, there might be a management agent that can request a shutdown or a reset, either gracefully through cooperation with the OSPM, or forcibly.
  - Virtualized systems might have a privileged agent, like a Hypervisor, that can request a shutdown or a reset, either gracefully through cooperation with the OSPM of the virtual machines, or forcibly.
  - Systems that deploy multiple Operating Systems running on different PE clusters within the same System-on-Chip might have a privileged agent. The privileged agent can request a shutdown or a reset, either gracefully through cooperation with the OSPM of the operating system entities in the different PE clusters, or forcibly in exceptional scenarios.
- The OSPM, which might receive notifications for a graceful shutdown request.

An agent can request the system to forcibly shut down or reset. The platform responds by performing the action that has been requested and then sending informational notifications to any remaining active agents in the system who have subscribed to the notification. An agent can also request a graceful shutdown or reset. In this case, the platform might send notifications to any subscribing OSPM agent, which can, in turn, initiate the requested action. To this end, the protocol allows an agent to request notifications of system power state transition requests generated by other agents. Table 10 describes the expected behavior for the various operations that are provided by this interface, depending on the calling agent. When a system has multiple agents of the same type, coordination between such agents of the same type is driven by an implementation defined policy.

Table 10: System power management operations, and expected responses depending on type of agent

Operation	Type of agent	Response
Request a forceful power state transition	OSPM	If the PE that the agent runs on supports PSCI, deny the request as NOT_SUPPORTED, as the calling agent is not in Secure or Root world. Otherwise, shutdown or reset as requested and send notifications.
	PSCI implementation on application processor	Shutdown or reset as requested and send notifications.
	Management agent or privileged agent	Shutdown or reset as requested and send notifications.
Request a graceful power state transition	OSPM	If the PE that the agent runs on supports PSCI, deny the request as NOT_SUPPORTED, as the calling agent is not in Secure or Root world. Otherwise allow the request and send notifications to other subscribing agents.
	PSCI implementation on application processor	Allow the request and send notifications to other subscribing agents.
	Management agent or privileged agent	Allow the request and send notifications to other subscribing OSPM agents.
Request for notification of power state transition requests	OSPM	Allow, as this agent will initiate a shutdown or reset in response to the notification.
	Management agent or privileged agent	Allow, to enable the management or privileged agent to confirm that the OSPM has requested shutdown or reset.
	PSCI	Deny. NOT_SUPPORTED, because it is not required to handle notifications.

When the PSCI agent initiates a system power state request, the corresponding notification is not sent to OSPM, since PSCI is deemed to act on behalf of OSPM. In general, notifications of system power state transitions are not propagated to the agent that requests the transition. The only exception is when a management agent requests a graceful system power state transition. In this case the management agent requesting the transition might need to confirm that the request has been acted upon by OSPM. A forceful system power state notification is sent to the management agent as a confirmation when the PSCI agent has requested a system power state transition on behalf of OSPM.

The protocol supports four kinds of system transitions:

- System powerup or shutdown.
- System suspend, as defined in PSCI for SYSTEM_SUSPEND, which is essentially a low-power system state. An example of a system suspend state is the suspend to RAM scenario that is analogous to S3 in ACPI.
- Architectural system resets, which are resets that are defined by this specification. These resets include system cold reset and system warm reset.
- Vendor defined transitions.

A system cold reset is equivalent to power cycling the system. All components in the system are powered down or held in reset. Components that are involved in the system boot are powered up or released from reset. In this context, the term cold boot refers to the expected boot flow after the first application of power to the system.

A system warm reset is one that preserves all memory that is visible to application processors. Like cold reset, all components in the system, except those involved in the provision of system memory to application processors, are powered down or held in reset. This definition of system memory does not extend to caches or to memory mapped I/O. As in the cold reset case, only those components that are involved in a system boot are powered up or released from reset.

System suspends could be of varying depths and wake latencies. Some suspend states could involve relatively large wake latencies, for example, suspend-to-RAM or SYSTEM_SUSPEND. Other suspend states, such as S0 idle states, could involve much lower wake latencies.

The view of the system that is affected by a system power state transition depends on the target segment and type of system being implemented. In some implementations, the system that is being powered down includes all the agents that can use this interface, as well as the platform controller that implements it. In this case this protocol is said to have a full-system view. However, for some platform implementations, the platform controller that implements this SCMI protocol might be in a dedicated always-on domain, such that it is not included in the system power transitions. In this case, this protocol is said to have an OSPM-system view, and the system power state transitions only affect those parts of the system that the OSPM controls. These parts are collectively called the OSPM world. In this latter kind of system, if an agent requests a system shutdown, the platform controller remains powered, so that it can service further commands, for example, a command to power up the system.

Table 11 describes the expected behavior for the various forcible (non-graceful) operations that are provided by this interface, depending on the calling agent and the view of the system implemented.

Table 11: System power management forcible operations, and expected responses depending on view

System view	Operation (forcible)	Calling Agent	Expected behavior
OSPM	Shutdown or system suspend	PSCI on behalf of the OSPM	The OSPM world is shut down or suspended. System power state notifications to other agents are sent at the point at which it is possible to request a system power up.
		Management agent or privileged agent	Message returns at the point at which it is possible to request a system power up.
	Reset	PSCI on behalf of the OSPM	The OSPM world is reset. System power state notifications to other agents are sent when it is possible to request forcible system shutdown or reset.
		Management agent or privileged agent	The OSPM world is reset. The message returns when it is possible to request forcible system shutdown or reset.
	Power-up	PSCI on behalf of the OSPM	Not supported.
		Management agent or privileged agent	The OSPM world is powered up. The message returns at the point at which forcible system power state requests are possible.
	Get system power state	PSCI on behalf of the OSPM	Not supported.
		Management agent or privileged agent	Message returns system power state of OSPM world.

System view	Operation (forcible)	Calling Agent	Expected behavior
Full	Shutdown or suspend	PSCI on behalf of the OSPM	Whole system – or the full-system world – is shut down or suspended. System power state notifications to other agents are sent at the point at which PSCI makes its request.
		Management agent or privileged agent	Whole system – or the full-system world – is shut down or suspended. Notifications in this case are not required.
	Reset	PSCI on behalf of the OSPM	System is Reset. System power state notifications to other agents are sent at the point at which PSCI makes its request.
		Management agent or privileged agent	System is Reset. Notifications in this case are not required.
	Power up or get system state	PSCI on behalf of the OSPM	Not supported.
		Management agent or privileged agent	Not supported.

In both full and OSPM-system view implementations, the behavior towards a PSCI or an OSPM agent remains unchanged. The change in behavior is only visible to an external agent, such as a management agent or privileged agent. Commands to power up or get system state are only present in systems that implement the OSPM system view.

3.4.2 Commands

3.4.2.1 PROTOCOL_VERSION

On success, this command returns the version of this protocol. For this version of the specification, the value returned must be `0x20001`, which corresponds to version 2.1.

message_id: `0x0`
protocol_id: `0x12`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this must be `0x20001`.

3.4.2.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: 0x10
protocol_id: 0x12
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.4.2.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: 0x1
protocol_id: 0x12
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 attributes	**Bits[31:0]** Reserved, must be zero.

3.4.2.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

message_id: 0x2
protocol_id: 0x12
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is invalid or not provided by this platform implementation. • NOT_SUPPORTED: when message_id is set to the SYSTEM_POWER_STATE_NOTIFY command identifier and notifications are not supported. See Section 3.1.4 for more status code definitions.
uint32 attributes	Flags associated with a specific command in the protocol. If message_id is for SYSTEM_POWER_STATE_SET, the attributes have the following format: **Bit[31]** System warm reset support. • Set to 1 if system warm reset is supported. • Set to 0 if system warm reset is not supported. **Bit[30]** System suspend support. • Set to 1 if system suspend is supported. • Set to 0 if system suspend is not supported. **Bits[29:0]** Reserved, must be zero. For all values of message_id, this value is zero.

3.4.2.5 SYSTEM_POWER_STATE_SET

This command is used to power down or reset the system.

System power-up must only be available to agents other than a PSCI implementation on systems that implement OSPM system view, as discussed in the Section 3.4.1.

message_id: `0x3`
protocol_id: `0x12`
This command is mandatory.

Parameters

Name	Description
uint32 flags	This parameter has the following format: **Bits[31:1]** Reserved, must be zero. **Bit[0]** Graceful request. This flag is ignored for power up requests. • Set to 1 if the request is a graceful request. • Set to 0 if the request is a forceful request.

uint32 system_state	Can be one of: 0x0 System shutdown. 0x1 System cold reset. 0x2 System warm reset. 0x3 System power-up. 0x4 System suspend. 0x5 – 0x7FFFFFFF Reserved, must not be used. 0x80000000 – 0xFFFFFFFF Might be used for vendor-defined implementations of system power state. These can include additional parameters. The prototype for vendor-defined calls is beyond the scope of this specification.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • INVALID_PARAMETERS: if the requested power state is not valid. • NOT_SUPPORTED: if the requested state is not supported for the calling agent. • DENIED: for system suspend requests when there are application processors, other than the caller, in a running or idle state. See Section 3.1.4 for more status code definitions.

3.4.2.6 SYSTEM_POWER_STATE_GET

This command must only be available to agents other than a PSCI implementation on systems that implement OSPM view, as discussed in Section 3.4.1. The command is to get the power state of the system.

message_id: 0x4
protocol_id: 0x12
This command is mandatory in an OSPM view implementation.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 system_state	Can be one of: 0x0 – System shutdown. 0x3 – System power-up. 0x4 – System suspend. 0x5 – 0x7FFFFFFF Reserved, must not be used. 0x80000000 – 0xFFFFFFFF Available for vendor-defined states.

3.4.2.7 SYSTEM_POWER_STATE_NOTIFY

This command is used to request notification of system power state requests. This command might be used:

- By the OSPM to receive notifications of graceful system power state requests.
- By a management agent or a privileged agent to be notified that the OSPM requested a forceful transition.

On initial boot of an agent, these notifications must be disabled by default to that agent.

message_id: 0x5
protocol_id: 0x12
This command is mandatory in an OSPM view implementation.

Parameters

Name	Description
uint32 notify_enable	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Notify enable: • If this value is set to 0, the platform does not send any SYSTEM_POWER_STATE_NOTIFIER messages to the calling agent. • If this value is set to 1, the platform does send SYSTEM_POWER_STATE_NOTIFIER messages commands to the calling agent. See Section 3.4.3.1 for details about SYSTEM_POWER_STATE_NOTIFIER notifications.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS • NOT_SUPPORTED: if notifications are not supported or available to the calling agent. • INVALID_PARAMETERS: if notify_enable specifies invalid or impermissible values. See Section 3.1.4 for more status code definitions.

3.4.3 Notifications

3.4.3.1 SYSTEM_POWER_STATE_NOTIFIER

If an agent has registered for system power state notifications with SYSTEM_POWER_STATE_NOTIFY, the platform sends this notification to the agent. Typically, the agent is either:

- The OSPM that initiates a system power state transition in response to this notification. The OSPM needs this notification to become aware that a remote entity such as the management agent or the privileged agent is requesting a graceful power state transition.
- A management agent or a privileged agent that initiated a graceful power state transition and is waiting for the OSPM to perform a power state transition in response. The management agent or privileged agent needs this notification to confirm that the platform controller has successfully received the power state transition request from the PSCI agent, or from the OSPM for non-PSCI compliant systems.

message_id: 0x0
protocol_id: 0x12
This command is optional.

Parameters

Name	Description

uint32 agent_id	Identifier for the agent that caused the system power state transition.
uint32 flags	This parameter has the following format: **Bits[31:1]** Reserved, must be zero. **Bit[0]** Graceful request. • Set to 1 if the notification indicates that a system power state transition has been gracefully requested. • Set to 0 if the notification indicates that a system power state has been forcibly requested. When sent to the management agent, it could also indicate that the PSCI agent has requested a system power state transition in response to an initial graceful request from the management agent.

uint32 system_state	System power state that the system has transitioned to, or which has been requested.
	Can be one of:
	`0x0` System shutdown.
	`0x1` System cold reset.
	`0x2` System warm reset.
	`0x3` System power-up.
	`0x4` System suspend.
	`0x5` – `0x7FFFFFFF`
	Reserved, must not be used.
	`0x80000000` – `0xFFFFFFFF`
	Available for vendor-defined implementations of system power state. These can include additional parameters. The prototype for vendor-defined call is beyond the scope of this specification.

uint32 timeout	This field is valid only when both the following conditions are satisfied: • the system_state field is system shutdown (`0x0`), and • Bit[0] of flags field is set to 1 (graceful request). This field indicates the platform-imposed timeout, specified in milliseconds, enforced from the point at which the platform sends a graceful system shutdown notification to OSPM. It indicates how long the platform waits for OSPM to gracefully shutdown the system. On timeout expiry, if the platform has not received the system shutdown request from the PSCI agent, or from OSPM for non-PSCI compliant systems, the platform can proceed to forcefully shutdown the system. If the platform does not impose any timeout, then this field is unused and must be set to zero.

3.5 Performance domain management protocol

This protocol is intended for the performance management of groups of devices or APs that run in the same performance domain. The protocol allows an agent to control the performance levels, and the performance limits of a domain. The protocol also allows an agent to configure the Quality of Service (QoS) parameters of a domain.

Performance domains must not be confused with power domains. A performance domain is defined as a set of devices which run at the same performance level or QoS configuration. For example, a set of CPUs that share a voltage rail, and have a common frequency control, can be represented as a performance domain.

QoS configurations are intended to help the platform determine how performance, power budgets and other shared resources are allocated in constrained systems. The specification uses the term QoS capabilities to describe the QoS configurations which the platform supports. Performance domains can describe and allow the configuration of QoS capabilities which impact performance.

The protocol allows a domain to implement either QoS controls, performance controls, or both. The commands in this protocol provide functionality to:

- Describe and negotiate the protocol version.
- Describe attribute flags of the protocol, messages, and performance domains.
- Describe QoS capabilities.
- Read and set performance levels and limits of performance domains.
- Read and set QoS configurations for QoS capabilities.
- Return the list of performance levels supported by a performance domain, and the properties of each performance level.
- Optionally return statistics on residency and usage count of a performance level in performance domains.

3.5.1 Performance domain management protocol background

The performance domain management protocol command set operates on an abstract integer performance scale of performance levels. The implementation can choose what this scale represents. For example, in some systems, the values in the scale might represent actual frequencies, while in others they might represent a percentage of the maximum performance of the domain. In all cases, the scale must be linear, meaning that a value of 2X delivers twice the performance as compared to a value of X.

Although this protocol uses an abstract scale to represent performance levels, the underlying implementation only provides a discrete set of performance levels.

Protocol commands take integer identifiers to describe which performance domain a given command applies to. The identifiers are sequential and start from 0.

In a multi-agent system, an agent can request the platform to set performance levels and limits for any domain. Such requests are allowed only if permissions are granted through the domain's attribute flags. Agents are also allowed to read performance data or register for notifications issued on performance changes. The platform is responsible for the resolution of levels and limits when multiple agents send simultaneous requests on the same performance domain.

Similarly, in a multi-agent system any agent can set QoS configurations for supported QoS capabilities of any domain. However, such requests are allowed only if the agent is granted access to configure the QoS as expressed through the domain's attribute flags. The platform is responsible for the resolution of QoS configurations when multiple agents send simultaneous requests on the same performance domain.

A performance domain can be characterized by four distinct performance levels that can be advertised by the platform. These distinct levels are described in Table 12. The performance domain can support additional performance levels.

Table 12: Performance Domain Levels with Special Significance

Performance Level	Description
Highest Performance	This is the theoretical maximum performance level of the domain.
Sustained Performance	This is the maximum performance level that the domain can sustain under normal operating conditions. The sustained performance level can account for external constraints like power budgeting and thermal mitigation, which were taken into account while defining the boundary of normal operating conditions. In exceptional circumstances, such as thermal runaway, the platform might not be able to guarantee this level. The exact definition of “normal operating conditions” is platform specific and out of the scope of this specification. All performance domains should be able to maintain their sustained performance level simultaneously under normal operating conditions.
Guaranteed Performance	This is the maximum performance level that the platform can guarantee under all known internal and external constraints, including power and thermal constraints. All performance domains should be able to run at their guaranteed performance levels concurrently. The guaranteed performance level is at least equal to the lowest performance level and does not exceed the sustained performance level.
Lowest Performance	This is the lowest performance level supported by the domain.

3.5.2 Power Cost

Each performance level has an associated power cost. If the performance domain includes APs, the power cost indicates the power consumption attributed to each AP in the performance domain. In all other cases, the power cost indicates the total power consumed by the performance domain when the domain is run at the given performance level. The performance domain protocol provides a command to discover performance levels and their associated power cost. The power cost can be expressed in microwatts (uW), milliwatts (mW) or in an abstract scale. Vendors are not obliged to reveal power costs, but if power costs are reported then a linear scale should be used.

3.5.3 Level Indexing Mode

Certain platforms use IMPLEMENTATION DEFINED indices to identify performance levels. Level Indexing Mode is used to describe such platform behavior. The level indices associated with performance levels are neither guaranteed to be contiguous nor required to be on a linear scale. Section 3.5.6.5 specifies how Level Indexing Mode is identified.

All commands which utilize performance level as a parameter need to specify the corresponding level index instead of the performance level when Level Indexing Mode is used. Failure to do so might result in unexpected platform behavior.

3.5.4 Quality of Service (QoS)

Performance domains might allow an agent to configure QoS for a domain. The specification uses the term QoS capabilities to identify controllable QoS parameters. This enables an agent to express domain specific quality-of-service requirements to the platform. QoS affects platform policy which can typically, but not exclusively, affect allocation of resources, whether shared or private, to a domain. The exhaustive list of platform policies impacted is IMPLEMENTATION DEFINED and outside the scope of this specification.

QoS capabilities are described by a Type and Subtype tuple:

- QoS Type identifies a class of QoS requirements. QoS Types are represented by a bitmap, where each bit that is set to 1 indicates a QoS Type. The specification allows architected and OEM defined ranges of QoS Types. See Table 13 and Section 3.5.4.1 for bitmap offsets and details on architected QoS Types. OEM defined QoS Types are outside the scope of this specification. The QoS Types supported for an agent are specified by the bitmap

returned in the capability_type field of the PERFORMANCE_DOMAIN_ATTRIBUTES command, as specified in section Section 3.5.6.5.

- The QoS Subtype refers to a specific implementation of a QoS Type, or a platform policy impacted by the QoS Type. The Subtype is optional, and its definition depends on the Type, as shown in Table 13. QoS Subtypes are represented by a bitmap, where each bit that is set to 1 represents a QoS Subtype. Multiple Subtypes can be implemented concurrently for the same Type. The specification allows architected and OEM defined ranges of QoS Subtypes. See Table 13 and Section 3.5.4.2 for bitmap offsets and details on architected QoS Subtypes. OEM defined QoS Subtypes are outside the scope of this specification. The QoS Subtypes supported for an agent are specified by the bitmap returned in the capability_subtype field of the PERFORMANCE_QOS_CAPABILITY_SUBTYPES command, as specified in section Section 3.5.6.6.

Table 13: Architected QoS capability associations

Architected Type	Bit offset in QoS Type Bitmap	Architected Subtype	Bit offset in QoS Subtype Bitmap
Relative Priority	0	Boost	0
		Throttle	1
		Boost and Throttle	2
Proportional Priority	1	Boost	0
		Throttle	1
		Boost and Throttle	2
Efficiency Defined Performance (EDP)	2	8-bit EDP	0
Operational mode	3	-	

Note

All other bit offsets in the architected QoS capability range are reserved for future use.

3.5.4.1 Architected QoS Types

The specification allows the following architected QoS Types:

- **Relative Priority**
  The priority of a domain relative to its sibling domains. Sibling domains are defined as domains which have the same parent. The parent identifier of a domain is provided by the PERFORMANCE_DOMAIN_ATTRIBUTES command, as specified in Section 3.5.6.5. Relative priorities are expressed using unsigned integer values, starting at 0. The qos_attribute_1 field of the PERFORMANCE_QOS_ATTRIBUTES command, as specified in Section 3.5.6.7, returns the maximum supported priority value N. If an agent configures a priority greater than N, the platform reduces it to N. Priorities are ordered [0…N], where a higher value indicates a lower priority. However, the relative priority identifiers are not scaled with respect to each other. This means 2X is not half the priority of X. If qos_attribute_1 is set to 0, all priority values are considered valid, though returning 0 is not recommended. Priority value 0xFFFFFFFF is reserved and must not be used. The same qos_attribute_1 value must be returned for all sibling domains.

- **Proportional Priority**
  The priority of a domain as a proportion of the sum of priorities across all its sibling domains, including the domain itself. Sibling domains are defined as domains which have the same parent. The parent identifier of a domain is provided by the PERFORMANCE_DOMAIN_ATTRIBUTES command, as specified in Section 3.5.6.5. Proportional priorities are expressed using unsigned integer values, starting at 1. A higher value indicates

a higher priority such that a value of 2X indicates double the priority of X. The qos_attribute_1 field of the PERFORMANCE_QOS_ATTRIBUTES command, as specified in Section 3.5.6.7, returns the maximum priority value, N, that can be set for the domain. Priorities are ordered [0…N], where a higher value indicates a higher proportional priority. If an agent configures a priority value greater than N, the platform reduces it to N. If the qos_attribute_1 field is set to 0, any priority value can be used for the domain. Priority values 0x0 and 0xFFFFFFFF are reserved and must not be used. The same The same qos_attribute_1 value must be returned for all the sibling domains.

—— **Note** ——————————

Proportional priority indicates a priority weight, with a higher value signifying a higher priority. In contrast, relative priority indicates an order of importance, with a lower value signifying a higher priority.
If the platform supports multiple QoS Priority Types, agents must select a single QoS Priority Type to use across sibling domains. Failure to adhere to this rule might result in unpredicatable behavior.

————————————————————

- **Efficiency Defined Performance (EDP)**
  EDP allows the agent to indicate the relative importance of energy efficiency in determining a domain's performance. This is an input into the platform's performance policies. Higher EDP values indicate an agent's preference towards greater energy efficiency. Lower values indicate a higher importance of performance, even at the expense of energy efficiency. EDP values are expressed using sequential unsigned integer values, starting at 0. The qos_attribute_1 field of the PERFORMANCE_QOS_ATTRIBUTES command, as specified in Section 3.5.6.7, returns the maximum EDP value N supported by the platform. The maximum possible value of N is governed by the QoS Subtype. If qos_attribute_1 is set to 0, the behaviour is QoS Subtype specific and described in Section 3.5.4.2. Valid EDP values are ordered [0…N]. EDP values across domains are not strictly comparable, since the energy efficiency sensitivity of platform policies might differ between domains. Therefore, EDP values are not scaled with respect to each other. This means 2X does not indicate twice the energy efficiency of X.

- **Operational Mode**
  Represents a specific QoS operational mode for the domain. The number of operational modes and their interpretation is IMPLEMENTATION DEFINED and outside the scope of this specification. Operational modes are expressed using sequential unsigned integer values, starting at 0. The qos_attribute_1 field of the PERFORMANCE_QOS_ATTRIBUTES command, as specified in Section 3.5.6.7, returns the maximum value, N, representing an operational mode. Valid QoS mode configurations fall in the interval [0…N].

  —— **Note** ——————————

  This Type is experimental and may be removed in future versions of the specification based on feedback.

  ————————————————————

3.5.4.2 Architected QoS Subtypes

The specification allows the following range of architected QoS Subtypes:

- **Boost**
  This QoS Subtype can only be associated with a QoS Priority Type. It represents the relative priority of a domain to be allocated additional performance above the domain's sustained performance level. A performance domain can only be considered for additional performance above its sustained performance level if all other active performance domains are operating at either their sustained performance level, or their requested performance levels, whichever is lower. A higher priority domain is allocated additional performance boost above its sustained performance level before a lower priority domain.

- **Throttle**
  This QoS Subtype can only be associated with a QoS Priority Type. It represents the priority of a domain to be throttled below its sustained performance level. A domain can only be considered for throttling below its sustained performance level if all other active performance domains are operating at no greater than their sustained performance level or the requested performance level, whichever is lower. A higher priority domain is throttled after a lower priority domain.

- **Boost and Throttle**
  This QoS Subtype can only be associated with a QoS Priority Type. It represents a single QoS control for both boost and throttle policies. The boost and throttle policies are specified under the Boost and the Throttle QoS Subtypes. If an agent chooses to use this QoS Subtype, it must not use either the Boost Subtype or the Throttle SubType on the domain and its sibling domains. Failure to adhere to this rule might result in unpredictable behaviour.

- **8-bit EDP**
  This QoS Subtype is associated with the QoS EDP Type. It indicates an agent's preference to balance performance and energy efficiency for the domain. EDP values from 0 to 255 are supported. Higher values indicate preference for higher energy efficiency even at the expense of performance. 255 is the highest value that can be returned in the qos_attribute_1 field of the PERFORMANCE_QOS_ATTRIBUTES command, as specified in Section 3.5.6.7. A value of 0 in the qos_attribute_1 field implies the entire range from 0 to 255 is supported. FastChannels for this Subtype must be exactly 8-bits wide.

3.5.5 FastChannels

This section describes the properties of FastChannels for Performance Domain Management Protocol.

- The following commands are supported over FastChannels:
  - PERFORMANCE_LIMITS_SET
  - PERFORMANCE_LIMITS_GET
  - PERFORMANCE_LEVEL_SET
  - PERFORMANCE_LEVEL_GET
  - REDUCE_SUSTAINED_PERFORMANCE_LEVEL
  - PERFORMANCE_QOS_CONFIG_SET
  - PERFORMANCE_QOS_CONFIG_GET
- A FastChannel needs to be unique for any combination of performance domain and performance domain management command. It is not necessary for every performance domain or every Performance Domain Management Command to support a FastChannel.
- FastChannels are discoverable via the PERFORMANCE_DESCRIBE_FASTCHANNEL command.
- Doorbell is not supported for PERFORMANCE_LEVEL_GET and PERFORMANCE_LIMITS_GET commands. If FastChannels are implemented for these commands, the last known valid performance level or performance limits must always be available over the FastChannel without a doorbell trigger. This property reduces complexity due to latency considerations between doorbell trigger and the availability of return values over the FastChannel. For all other commands, Doorbell support is optional.

For more details on FastChannels, see Section 4.3

3.5.5.1 Payload Requirements

The payload of a FastChannel must only contain the message-specific parameters and exclude the domain_id. Since a FastChannel is domain_id and message_id specific, the domain_id or any other channel-specific and message-specific headers do not need to be included while using a FastChannel. For example, the payload of the PERFORMANCE_LEVEL_SET message should be 'uint32 performance_level', while the payload of the PERFORMANCE_QOS_CONFIG_SET command for the 8-bit EDP Subtype should be 'uint8 qos_value'.

3.5.6 Commands

3.5.6.1 PROTOCOL_VERSION

On success, this command returns the version of this protocol. For this version of the specification, the value returned must be `0x40001`, which corresponds to version 4.1.

message_id: 0x0
protocol_id: 0x13
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this must be 0x40001.

3.5.6.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: 0x10
protocol_id: 0x13
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.5.6.3 PROTOCOL_ATTRIBUTES

This command returns the attributes associated with this protocol.

message_id: 0x1
protocol_id: 0x13
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.

uint32 attributes	**Bits[31:18]** Reserved, must be zero. **Bits[17:16]** Power Unit: • Set to 2 if the power consumption of performance levels is expressed in uW. • Set to 1 if the power consumption of performance levels is expressed in mW. • Set to 0 if the power consumption of performance levels is expressed in an abstract linear scale. • All other values are reserved and must not be used. **Bits[15:0]** Number of performance domains.
uint32 statistics_address_low	The lower 32 bits of the physical address where the statistics shared memory region is located. This value should be 64-bit aligned. The address must be in the memory map of the calling agent. If the statistics_len field is 0, then this field is invalid and must be ignored. The statistics shared memory region is described in Section 3.5.9.
uint32 statistics_address_high	The upper 32 bit of the physical address where the shared memory region is located. The address must be in the memory map of the calling agent. If the statistics_len field is 0, then this field is invalid and must be ignored. The statistics shared memory region is described in Section 3.5.9.
uint32 statistics_len	The length in bytes of the shared memory region. A value of 0 in this field indicates that the platform doesn't support the statistics shared memory region.

Note: Statistics capability has been marked for deprecation and is being replaced by system telemetry.

3.5.6.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

This command can be used to enquire if performance level or limit change notifications are supported by the platform. This is achieved by passing message identifiers for the PERFORMANCE_NOTIFY_LEVEL or PERFORMANCE_NOTIFY_LIMITS messages to the call. The platform then returns a status code of NOT_FOUND to indicate that notifications are not implemented, or that they are not available to the calling agent. The notification commands are described in sections Section 3.5.6.16 and Section 3.5.6.17. This command can also be used to discover if FastChannels are supported for a command specified by message_id.

message_id: `0x2`
protocol_id: `0x13`
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is invalid or not provided by this platform implementation. See Section 3.1.4 for more status code definitions.
uint32 attributes	Flags associated with a specific command in the protocol. **Bits[31:1]** Reserved, must be zero. **Bit[0]** FastChannel Support. • Set to 1 if there is at least one dedicated FastChannel available for this message. • Set to 0 if there are no FastChannels available for this message.

3.5.6.5 PERFORMANCE_DOMAIN_ATTRIBUTES

This command returns attributes that are specific to a given domain.

message_id: `0x3`
protocol_id: `0x13`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid performance domain attributes are found. • NOT_FOUND: if domain_id does not point to a valid domain. See Section 3.1.4 for more status code definitions.

uint32 attributes	**Bit[31]** Can set limits. • Set to 1 if calling agent is allowed to set the performance limits on the domain. • Set to 0 if a calling agent is not allowed to set limits on the performance limits on the domain. **Bit[30]** Can set performance level. • Set to 1 if calling agent is allowed to set performance of a domain. • Set to 0 if a calling agent is not allowed to set the performance of a domain. **Bit[29]** Performance limits change notifications support. • Set to 1 if performance limits change notifications are supported for this domain. • Set to 0 if performance limits change notifications are not supported for this domain. **Bit[28]** Performance level change notifications support. • Set to 1 if performance level change notifications are supported for this domain. • Set to 0 if performance level change notifications are not supported for this domain. **Bit[27]** FastChannel Support. • Set to 1 if there is at least one FastChannel available for this domain • Set to 0 if there are no FastChannels available for this domain. **Bit[26]** Extended performance domain name. • If set to 1, the performance domain name is greater than 16 bytes. The extended performance domain name is provided by PERFORMANCE_DOMAIN_NAME_GET command which is described in Section 3.5.6.19. • If set to 0, extended performance domain name is not supported. **Bit[25]** Level Indexing Mode • If set to 1, it indicates that the domain uses Level Indexing Mode. All commands which utilize performance level as a parameter need to specify the corresponding level index instead of the performance level when Level Indexing Mode is used. • If set to 0, Level Indexing Mode is not used. **Bit[24]** Asynchronous QoS configuration support • If set to 1, all QoS capabilities of this domain can be configured asynchronously. • If set to 0, the QoS capabilities of this domain can only be configured synchronously. **Bit[23]** QoS only domain • If set to 1, this domain does not support performance levels and their visibility to the agent. If this bit is set to 1, Bits[31:28] must be 0. • If set to 0, this domain supports agent visible performance levels. Bits[31:28] specify the performance controls available.

	Bits[22:0] Reserved and set to zero.
uint32 rate_limit	**Bits[31:20]** Reserved and set to zero. **Bits[19:0]** Rate Limit in microseconds, indicating the minimum time required between successive requests. A value of 0 indicates that this field is not supported by the platform. This field does not apply to FastChannels.
uint32 sustained_freq	Base frequency corresponding to the sustained performance level. Expressed in units of kHz. This field can be set to 0 by the platform, if Bit[23] of the attributes field is set to 1.
uint32 sustained_perf_level	The performance level value that corresponds to the maximum sustained performance level for the domain. This field can be set to 0 by the platform, if Bit[23] of the attributes field is set to 1. This field is a platform specific value, and is unchanged when an agent voluntarily reduces the domain's sustained performance requirement through the REDUCE_SUSTAINED_PERFORMANCE_LEVEL command, as specified in section Section 3.5.6.13
uint8 name[16]	Null terminated ASCII string of up to 16 bytes in length describing a domain name. When Bit[26] of attributes field is set to 1, this field contains the lower 15 bytes of the NULL terminated performance domain name.
uint32 guaranteed_perf_level	The performance level value that corresponds to the guaranteed performance delivered by the platform. This field can be set to 0 if the platform does not wish to specify the guaranteed performance level.
uint32 qos_capability_types	Bitmap of supported QoS capability Types If this field is set to 0, QoS is not supported by the domain. This field cannot be 0 if Bit[23] of the attributes field is set to 1. **Bits[31:24]** Reserved, must be zero. **Bits[23:16]** OEM QoS capability Type bitmap where each bit indicates support for an OEM capability Type. This field may have multiple bits set. **Bits[15:8]** Reserved, must be zero. **Bits[7:0]** Architected QoS capability Type bitmap where each bit indicates support for an architected capability Type. The bit offsets and the Type descriptions are provided in Section 3.5.4.1 and Table 13. This field might have multiple bits set.
uint32 qos_parent_id	This field identifies the parent of the domain. This is useful in constructing a QoS topology when the current domain, identified by domain_id, is contained within a parent domain, identified by parent_id. A value of 0xFFFFFFFF implies that this domain is not contained within any parent domain. This field is valid for QoS capabilities only and can be ignored if the capability_types field is set to 0.

3.5.6.6 PERFORMANCE_QOS_CAPABILITY_SUBTYPES

This command returns the QoS capability Subtypes supported for a specified QoS capability Type.

message_id: 0xD
protocol_id: 0x13
This command is mandatory if QoS capabilities are supported by the platform.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.
uint32 capability_type	QoS capability Type bitmap. **Bits[31:16]** Reserved, must be zero. **Bit[15]** QoS capability Type range. • If set to 0, it indicates the architected range of QoS capability Types. • If set to 1, it indicates the OEM range of QoS capability Types **Bits[14:8]** Reserved, must be zero. **Bits[7:0]** QoS capability Type bitmap where each bit indicates a capability Type. Only one bit must be set, indicating the target QoS capability Type. The architected QoS capability Types and their bit offsets are provided in Section 3.5.4.1 and Table 13.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid QoS Subtypes are returned. • NOT_FOUND: if the domain_id parameter does not point to a valid domain or if the capability_type does not point to a valid capability Type of that domain. • INVALID_PARAMETERS: if the capability_type field has multiple Type bits set. See Section 3.1.4 for more status code definitions.
uint32 capability_subtypes	Supported QoS capability Subtypes bitmap. **Bits[31:24]** Reserved, must be zero. **Bit[23:16]** OEM QoS capability Subtype bitmap where each bit indicates a QoS capability Subtype. This field may have multiple bits set. **Bits[15:8]** Reserved, must be zero. **Bits[7:0]** Architected QoS capability Subtype bitmap where each bit indicates a QoS capability Subtype. The bit offsets and the Subtype description are provided in Table 13 and Section 3.5.4.2. This field may have multiple bits set.

3.5.6.7 PERFORMANCE_QOS_ATTRIBUTES

This command returns the name and attributes associated with a QoS capability.

message_id: 0xE
protocol_id: 0x13
This command is mandatory if QoS capabilities are supported by the platform.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.
uint32 capability	QoS capability, as described by Table 14. Only one QoS capability Type bit can be set. At most one QoS capability Subtype bit can be set.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid performance domain QoS attributes are found. • NOT_FOUND: if the domain_id parameter does not point to a valid domain or if the capability does not point to a valid capability of that domain. • INVALID_PARAMETERS: if the capability field has multiple Type or Subtype bits set. See Section 3.1.4 for more status code definitions.
uint32 qos_attribute_1	First attribute of the specified QoS capability of this domain. See Section 3.5.4.1 for attribute details of each QoS capability Type.
uint8 name[16]	Null-terminated ASCII string of up to 16 bytes in length representing the name of the QoS capability.

3.5.6.8 PERFORMANCE_DESCRIBE_LEVELS

This command allows the agent to ascertain the discrete performance levels that are supported by the platform, their respective power costs, transition latency, clock frequency and level index.

On success, the command returns an array that consists of several performance level entries, each of which describes an expected performance, power cost, transition latency, clock frequency and level index.

The performance levels returned by this call should be in numerically ascending order of performance level values. The power cost can be expressed in microwatts, milliwatts or on an abstract scale. How the numbers in that scale convert to the actual wattage is IMPLEMENTATION DEFINED, but the conversion must be linear, meaning that a power of 2X is twice the power of X. The size of the array depends on the number of return values that a given transport can support.

It might not be possible to return information for all performance levels with just one call. To solve this problem, the interface allows multiple calls, with a skip_index used to skip over performance levels which were returned by previous calls.

message_id: 0x4
protocol_id: 0x13
This command is mandatory if Bit [23] of the attributes field of the PERFORMANCE_DOMAIN_ATTRIBUTES command is set to 0 for any domain.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.

uint32 skip_index	Number of performance levels, in numerically ascending order of level values, to skip over before returning the first performance level in the return performance level array. For more information on how this field is used, refer to the pseudocode below. NOTE: This field is not the same as the level index of a performance level when Level Indexing Mode is used.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid performance levels are returned. • NOT_FOUND: if domain_id does not point to a valid domain. See Section 3.1.4 for more status code definitions.
uint32 num_levels	**Bits[31:16]** Number of remaining performance levels. **Bits[15:12]** Reserved, must be zero. **Bits[11:0]** Number of performance levels that are returned by this call.
{uint32, uint32, uint32, uint32, uint32} perf_levels[N]	Array of performance levels, in numeric ascending order, to be described. N is specified by Bits[11:0] of num_levels field. Each array entry is composed of five 32-bit words with the following format: **uint32 entry[0]** Performance level value. **uint32 entry[1]** Power cost. A value of zero indicates that the power cost is not reported by the platform. Refer to Section 3.5.2 for more details. **uint32 entry[2]** Attributes **Bits[31:16]** Reserved, must be zero. **Bits[15:0]** Worst-case transition latency in microseconds to move from any supported performance level to the level indicated by this entry in the array. **uint32 entry[3]** Indicative Frequency Clock frequency, in kHz, of the performance domain when operating at this performance level. When set to 0, it indicates that this field is not reported by the platform. **uint32 entry[4]** Level Index The level index for this performance level. If Bit[25] (Level Indexing Mode) of the attributes field returned by PERFORMANCE_DOMAIN_ATTRIBUTES command is set to 0, this field should be ignored by the agent. See Section 3.5.6.5 for more details.

The following pseudocode describes how the command can be used to discover information about every supported performance level for the performance domain:

```
uint16 skip_index = 0;
int32 status = 0;
struct number_of_perf_levels {
    uint perf_levels_array_len:12;
    uint reserved: 4;
```

```
    uint remaining:16;
} num_levels = {0,0,0};

struct perf_level_data {
    uint32 perf_value;
    uint32 power;
    uint16 transition_latency;
    uint16 reserved;
    uint32 indicative_freq;
    uint32 level_index;
};

struct perf_level_data perf_levels[];

do {
     invoke_PERFORMANCE_DESCRIBE_LEVELS (domain_id, skip_index,
     &status, &num_levels, perf_levels);
     if (status)
            goto clean_up_and_return;
     add_levels_to_database (domain_id, skip_index,
                             num_levels.perf_levels_array_len, perf_levels);
     skip_index += num_levels.perf_levels_array_len;

} while(num_levels.remaining);
```

3.5.6.9 PERFORMANCE_LIMITS_SET

This command allows the caller to set limits on the performance level of a domain. If Level Indexing Mode is used by the platform, the limits must be specified in level index instead of the corresponding performance level.

message_id: 0x5
protocol_id: 0x13
This command is mandatory if Bit [23] of the attributes field of the PERFORMANCE_DOMAIN_ATTRIBUTES command is set to 0 for any domain.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.
uint32 range_max	Maximum allowed performance level, or level index. If this field is set to 0, the platform ignores it and any pre-existing limit on maximum allowed performance level is left unchanged. Both range_max and range_min cannot be zero at the same time.
uint32 range_min	Minimum allowed performance level, or level index. If this field is set to 0, the platform ignores it and any pre-existing limit on minimum allowed performance level is left unchanged. Both range_max and range_min cannot be zero at the same time.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if the command successfully set the limits of operation. If setting a limit requires modifying the current performance level of the domain, the command can return before this change has been completed. However, the change in performance level must still take place. • NOT_FOUND: if the performance domain identified by domain_id does not exist. • OUT_OF_RANGE: if the limits set lie outside the highest and lowest performance levels that are described by PERFORMANCE_DESCRIBED_LEVELS, or if range_max and range_min are both zero. • DENIED: if the calling agent is not permitted to change the performance limits for the domain, as described by PERFORMANCE_DOMAIN_ATTRIBUTES. See Section 3.1.4 for more status code definitions.

3.5.6.10 PERFORMANCE_LIMITS_GET

This command allows the agent to ascertain the range of allowed performance levels, or level indices when Level Indexing Mode is used by the platform. The returned value reflects the currently set limits for the performance domain. These limits might have been set implicitly by the platform, or explicitly by a preceding call to PERFORMANCE_LIMIT_SET.

On success, the range return value provides the minimum and maximum allowed performance level, or level index.

message_id: `0x6`
protocol_id: `0x13`
This command is mandatory if Bit [23] of the attributes field of the PERFORMANCE_DOMAIN_ATTRIBUTES command is set to 0 for any domain.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the performance limits are returned successfully. • NOT_FOUND: if domain_id does not point to a valid domain. See Section 3.1.4 for more status code definitions.
uint32 range_max	Maximum allowed performance level, or level index.
uint32 range_min	Minimum allowed performance level, or level index.

3.5.6.11 PERFORMANCE_LEVEL_SET

This command allows the agent to set the performance level of a domain. If Level Indexing Mode is used, the limits must be specified in level index instead of the corresponding performance level. This command can return before the domain has transitioned to the required performance level. The platform simply has to acknowledge that it has received the command. The agent can register for performance level notifications to ascertain whether a performance transition has taken place. For further details, see Section 3.5.8.2.

message_id: 0x7
protocol_id: 0x13
This command is mandatory if Bit [23] of the attributes field of the PERFORMANCE_DOMAIN_ATTRIBUTES command is set to 0 for any domain.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.
uint32 performance_level	Requested performance level, or level index. If this field indicates performance level and if it is set to 0, platform policy determines the performance level of this domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the platform has accepted the command and scheduled it for processing. • NOT_FOUND: if the domain_id parameter does not point to a valid domain. • OUT_OF_RANGE: if the requested performance level is outside the currently allowed range. • DENIED: if the calling agent is not permitted to change the performance level for a domain, as described by PERFORMANCE_DOMAIN_ATTRIBUTES. See Section 3.1.4 for more status code definitions.

3.5.6.12 PERFORMANCE_LEVEL_GET

On success, this command returns the current performance level of a domain, or level index when Level Indexing Mode is used by the platform. Note the performance level that is returned by this command might be stale by the time the command completes, as a subsequent performance change might have been initiated in the meantime.

message_id: 0x8
protocol_id: 0x13
This command is mandatory if Bit [23] of the attributes field of the PERFORMANCE_DOMAIN_ATTRIBUTES command is set to 0 for any domain.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the performance level is returned successfully • NOT_FOUND: if domain_id does not point to a valid domain. See Section 3.1.4 for more status code definitions.
uint32 performance_level	Current performance level, or level index, of the domain.

3.5.6.13 REDUCE_SUSTAINED_PERFORMANCE_LEVEL

This command enables an agent to voluntarily request a reduction in the sustained performance level of a domain below the level specified by the sustained_perf_level field of the PERFORMANCE_DOMAIN_ATTRIBUTES command as specified in section Section 3.5.6.5. The platform provisions the reduced sustained performance level for the domain allowing to free up performance headroom to allocate to other domains. If Level Indexing Mode is used, this level must be specified in level index instead of the corresponding performance level. This command can return before the new sustained performance level has been set. The platform simply has to acknowledge that it has received the command. It is illegal to specify a sustained performance level which is higher than the platform specified sustained performance level. This command does not change the value returned in the sustained_perf_level field of the PERFORMANCE_DOMAIN_ATTRIBUTES command as specified in section Section 3.5.6.5, allowing the agent can revert back to the platform specified sustained performance level at any time.

message_id: `0xF`
protocol_id: `0x13`
This command is optional.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.
uint32 sustained_level	Requested sustained performance level, or level index.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the platform has accepted the command and scheduled it for processing. • NOT_FOUND: if the domain_id parameter does not point to a valid domain. • OUT_OF_RANGE: if the requested level is outside the currently allowed range. See Section 3.1.4 for more status code definitions.

3.5.6.14 PERFORMANCE_QOS_CONFIG_SET

This command allows the agent to set or reset the QoS configuration for a domain or group of domains, for a specified QoS capability.

message_id: `0x20`
protocol_id: `0x13`
This command is mandatory if QoS capabilities are supported by the platform.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.
uint32 capability	QoS capability, as described by Table 14. Only one QoS capability Type bit can be set. At most one QoS capability Subtype bit can be set.

uint32 flags	**Bits[31:5]** Reserved, must be zero. **Bit[4]** Platform QoS configuration reset • If set to 1, all QoS values for the specified QoS capability for all domains will be reset to the platform default. The domain_id and qos_value fields are ignored, if this flag is set to 1. • If set to 0, the QoS values for the specified QoS capability will not be reset to the platform default for all domains. **Bit[3]** Sibling domains QoS configuration reset. Sibling domains are defined as domains which have the same parent. • If set to 1, all QoS values for the specified QoS capability for this domain and its sibling domains will be reset to the platform default. The qos_value field is ignored, if this flag is set to 1. • If set to 0, the QoS values of the specified QoS capability will not be reset to the platform default for this domain and its sibling domains. **Bit[2]** Domain QoS configuration reset • If set to 1, the QoS value of the specified QoS capability of this domain will be reset to the platform default. The qos_value field is ignored if this flag is set to 1. • If set to 0, the QoS value of the specified QoS capability of this domain will not be reset to the platform default. **Bit[1]** Async flag • If set to 1, the specified QoS configuration must be set or reset asynchronously. – If the ignore delayed response Bit[0] is set to 0, the call is completed with the PERFORMANCE_QOS_CONFIG_COMPLETE message. – If the ignore delayed response Bit[0] is set to 1, the PERFORMANCE_QOS_CONFIG_COMPLETE response is not sent. A SUCCESS return code indicates that the platform has successfully queued this command. For details see Section 3.5.7.1. • Set to 0 if the QoS configuration must be set or reset synchronously. In this case the call will return when the action has been completed. **Bit[0]** Ignore delayed response • If the Async flag Bit[1] is set to 1 when this bit is set to 1, the platform does not send a PERFORMANCE_QOS_CONFIG_COMPLETE delayed response. • If the Async flag Bit[1] is set to 0, then this bit field is ignored by the platform.
uint32 qos_value	Requested QoS value for the specified QoS capability. This value is ignored if any of Bit[2], Bit[3], or Bit[4] in the flags field is set to 1.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if the QoS configuration was set or reset successfully for a synchronous request or if the command was successfully enqueued for an asynchronous request. • NOT_FOUND: if the domain_id parameter does not point to a valid domain or if the capability descriptor does not point to a valid QoS capability of this domain. • INVALID_PARAMETERS: – If the capability field has multiple Type or Subtype bits set, – If the flags parameter specifies invalid or illegal options, – If the requested QoS value is not supported. • DENIED: if the calling agent is not permitted to change the QoS configuration of the domain for the specified QoS capability. See Section 3.1.4 for more status code definitions.

The QoS capability descriptor has the following format:

Table 14: QoS capability descriptor

Field	Description
Bit [31]	QoS capability Type range. • If set to 0, it indicates the architected range of Types. • If set to 1, it indicates the OEM range of Types.
Bits [30:24]	Reserved, must be zero.
Bits [23:16]	QoS capability Type bitmap. Each bit indicates a capability Type. Only one bit must be set. The allocation of bits for architected Types is described in Table 13.
Bit [15]	QoS capability Subtype range. • If set to 0, it indicates the architected range of Subtypes. • If set to 1, it indicates the OEM range of Subtypes.
Bits [14:8]	Reserved, must be zero.
Bits [7:0]	QoS capability Subtype bitmap. Each bit indicates a capability Subtype. At most one bit can be set. The allocation of bits for architected Subtypes is described in Table 13.

3.5.6.15 PERFORMANCE_QOS_CONFIG_GET

This command allows an agent to obtain the current QoS configuration for the domain, associated with one of the domain's supported QoS capabilities.

Note

It is possible for the qos_value returned by this command to be stale by the time the command completes, as another QoS value change request could have been initiated and completed in the interim.

message_id: 0x21
protocol_id: 0x13
This command is mandatory if QoS capabilities are supported by the platform.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.
uint32 capability	QoS capability, as described by Table 14. Only one QoS capability Type bit can be set. At most one QoS capability Subtype bit can be set.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the QoS value was returned successfully. • NOT_FOUND: if the domain_id parameter does not point to a valid domain or if the capability does not point to a valid capability of that domain. • INVALID_PARAMETERS: if the capability field has multiple Type or Subtype bits set. See Section 3.1.4 for more status code definitions.
uint32 qos_value	The QoS configuration of the performance domain, associated with the given QoS capability.

3.5.6.16 PERFORMANCE_NOTIFY_LIMITS

This command allows the agent to request notifications from the platform for changes in the allowed maximum and minimum performance levels. These notifications are sent using the PERFORMANCE_LIMITS_CHANGED command which is described in Section 3.5.8.1.

If no domain supports limit notifications, the command can be omitted. The PROTOCOL_MESSAGE_ATTRIBUTES command, that is described in Section 3.5.6.5, can be used to determine whether this command is implemented.

On initial boot of an agent, by default, these notifications must be disabled from being sent to that agent.

message_id: 0x9
protocol_id: 0x13
This command is optional.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.

uint32 notify_enable	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Notify enable: • If this value is 0, the platform does not send any PERFORMANCE_LIMITS_CHANGED messages to the agent. • If this value is set to 1, the platform does send PERFORMANCE_LIMITS_CHANGED messages to the agent. See Section 3.5.8.1 for more details about PERFORMANCE_LIMITS_CHANGED notifications.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if domain_id does not point to a valid domain. • NOT_SUPPORTED: if notifications are not supported for the indicated performance domain. • INVALID_PARAMETERS: if notify_enable specifies values that are not legal or valid. See Section 3.1.4 for more status code definitions.

3.5.6.17 PERFORMANCE_NOTIFY_LEVEL

This command allows the agent to request notifications from the platform when the performance level for a domain changes in value. These notifications are sent using the PERFORMANCE_LEVEL_CHANGED command which is described in Section 3.5.8.2.

If no domains support level change notifications, the command can be omitted.

The PROTOCOL_MESSAGE_ATTRIBUTES command, that is described in Section 3.5.6.5, can be used to determine whether this command is implemented.

On initial boot of an agent, by default, these notifications must be disabled from being sent to that agent.

message_id: `0xA`
protocol_id: `0x13`
This command is optional.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.

uint32 notify_enable	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Notify enable: • If this value is 0, the platform does not send any PERFORMANCE_LEVEL_CHANGED notifications to the agent. • If this value is set to 1, the platform does send PERFORMANCE_LEVEL_CHANGED notifications to the agent. See Section 3.5.8.2 for more details about the PERFORMANCE_LEVEL_CHANGED notification.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if domain_id does not point to a valid domain. • NOT_SUPPORTED: if notifications are not supported for the indicated performance domain. • INVALID_PARAMETERS: if notify_enable specifies illegal or unimplemented options. See Section 3.1.4 for more status code definitions.

3.5.6.18 PERFORMANCE_DESCRIBE_FASTCHANNEL

This command allows the agent to discover the attributes of the FastChannel for the specified performance domain and the specified message.

The PERFORMANCE_DOMAIN_ATTRIBUTES command can be used to discover if a performance domain supports FastChannels. The PROTOCOL_MESSAGE_ATTRIBUTES command can be used to discover if a command, specified by message_id, supports FastChannels.

message_id: `0xB`
protocol_id: `0x13`
This command is optional.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain for which the FastChannel is allocated.
uint32 message_id	Message-id for which the FastChannel is allocated.
uint32 capability	Descriptor for the capability for which the FastChannel is allocated, as described by Table 14. Only a single QoS Type and a single Qos Subtype bit can be set. This field is set to 0 if the message_id does not use QoS capabilities.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: If a valid FastChannel is found. • NOT_FOUND: if domain_id does not point to a valid domain or message_id does not point to a valid message. • NOT_SUPPORTED: if FastChannel is not supported for this domain or this message. • INVALID_PARAMETERS: if the capability field has multiple Type or Subtype bits set. See Section 3.1.4 for more status code definitions.
uint32 attributes	**Bits[31:3]** Reserved. Should be zero in this version of the specification. **Bits[2:1]** Doorbell Register width. This field is only valid if Doorbell Support is set to 1. • If 0, then doorbell register is 8bits wide. • If 1, then doorbell register is 16bits wide. • If 2, then doorbell register is 32bits wide. • If 3, then doorbell register is 64bits wide. **Bit[0]** Doorbell Support. • If 0, then the FastChannel does not have a doorbell register. • If 1, then the FastChannel has a doorbell register.
uint32 rate_limit	**Bits[31:20]** Reserved and set to zero. **Bits[19:0]** Rate Limit in microseconds, indicating the minimum time required between successive requests. A value of 0 indicates that this field is not applicable or supported on the platform.
uint32 chan_addr_low	Lower 32 bits of the FastChannel address.
uint32 chan_addr_high	Higher 32 bits of the FastChannel address.
uint32 chan_size	Size of the FastChannel in bytes. The value of this field should be sufficient to accommodate the payload of the message this FastChannel is used for. For more details on payload requirements please refer Section 3.5.5.1.
uint32 doorbell_addr_low	Lower 32 bits of the doorbell address. This field is not used if doorbell is not supported.
uint32 doorbell_addr_high	Higher 32 bits of the doorbell address. This field is not used if doorbell is not supported.
uint32 doorbell_set_mask_low	Contains a mask of lower 32 bits to set when writing to the doorbell register. If the doorbell register width, n, is less than 32 bits, then only n lower bits are considered from this mask. This field is not used if doorbell is not supported.
uint32 doorbell_set_mask_high	Contains a mask of higher 32 bits to set when writing to the doorbell register. This field is only valid if the doorbell register width is 64 bits. This field is not used if doorbell is not supported.

uint32 doorbell_preserve_mask_low	Contains a mask of lower 32 bits to preserve when writing to the doorbell register. If the doorbell register width, n, is less than 32 bits, then only n lower bits are considered from this mask. This field is not used if doorbell is not supported.
uint32 doorbell_preserve_mask_high	Contains a mask of higher 32 bits to preserve when writing to the doorbell register. This field is only valid if the doorbell register width is 64 bits. This field is not used if doorbell is not supported.

Bits which are set neither in set_mask nor in preserve_mask are to be cleared.

3.5.6.19 PERFORMANCE_DOMAIN_NAME_GET

This command returns the extended name associated with a specific performance domain.

message_id: `0xC`
protocol_id: `0x13`
This command is mandatory only if extended performance domain name is supported.
Extended performance domain name is supported when Bit[26] of attributes field of PERFORMANCE_DOMAIN_ATTRIBUTES command is set to 1. Refer to Section 3.5.6.5 for more details.

Parameters

Name	Description
uint32 domain_id	Identifier for the performance domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid performance domain extended name is returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint8 name[64]	Null-terminated ASCII string of up to 64 bytes in length describing the performance domain extended name.

3.5.7 Delayed responses

3.5.7.1 PERFORMANCE_QOS_CONFIG_COMPLETE

If an agent has changed the QoS configuration for a QoS capability of a performance domain asynchronously through PERFORMANCE_QOS_CONFIG_SET, the platform sends this delayed response to the agent when the QoS setting request has been completed.

message_id: `0x20`
protocol_id: `0x13`
This command is optional.

Parameters

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if the QoS configuration was set or reset successfully. See Section 3.1.4 for more status code definitions.
uint32 domain_id	Identifier for the performance domain whose QoS configuration was set or reset. This is set to 0xFFFFFFFF if Bit[4] in the flags field is set to 1.
uint32 capability	QoS capability, as described by Table 14.
uint32 flags	**Bits[31:5]** Reserved, must be zero. **Bit[4]** Platform QoS configuration reset • If set to 1, all QoS values for the specified QoS capability for all domains have been reset to the platform default. • If set to 0, the QoS values for the specified QoS capability have not been reset to the platform default for all domains. **Bit[3]** Sibling domains QoS configuration reset. Sibling domains are defined as domains which have the same parent. • If set to 1, all QoS values for the specified QoS capability for this domain and its sibling domains have been reset to the platform default. • If set to 0, the QoS values of the specified QoS capability have not been reset to the platform default for this domain and its sibling domains. **Bit[2]** Domain QoS configuration reset • If set to 1, the QoS value of the specified QoS capability of this domain has been reset to the platform default. • If set to 0, the QoS value of the specified QoS capability of this domain has not been reset to the platform default. **Bits[1:0]** Reserved, must be zero.
uint32 qos_value	The QoS configuration that the domain has been set to, for the given QoS capability. This field should be ignored by the agent if any of Bit[2], Bit[3], or Bit[4] of the flags field is set to 1.

3.5.8 Notifications

3.5.8.1 PERFORMANCE_LIMITS_CHANGED

If an agent has registered for limit change notifications for the domain that is identified by domain_id, the platform sends this notification to the agent when the performance limits for that domain change.

message_id: `0x0`
protocol_id: `0x13`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier for the agent that caused the performance limit change.
uint32 domain_id	Identifier for the performance domain whose limit was changed.

uint32 range_max	Maximum allowed performance level, or level index when Level Indexing Mode is used.
uint32 range_min	Minimum allowed performance level, or level index when Level Indexing Mode is used.

3.5.8.2 PERFORMANCE_LEVEL_CHANGED

This notification is sent by the platform to an agent which has subscribed to receive performance level change notifications for the domain that is identified by domain_id. The platform sends this notification to a subscribing agent:

- when the performance level of the domain is changed by a different agent or entity in the system, including the platform itself.
- when the performance level of the domain is changed as a result of the subscribing agent issuing a PERFORMANCE_LEVEL_SET or PERFORMANCE_LIMITS_SET command.

The platform might autonomously change the performance level of the domain in order to apply thermal or power constraints. An external agent, such as a system management agent, can also request the platform to change the performance level. In each of these occurrences, the subscribing agent will be notified so that it can become aware of the change.

message_id: `0x1`
protocol_id: `0x13`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier for the agent that caused the performance level change.
uint32 domain_id	Identifier for the performance domain whose level was changed.
uint32 performance_level	The new performance level, or level index of the domain as a result of the change.

3.5.9 Performance domain statistics shared memory region

Note: Statistics capability has been marked for deprecation and is being replaced by system telemetry.

Optionally, the platform can provide a statistics memory region that is associated with the performance domain management protocol. Whether support is present is indicated by the PROTOCOL_ATTRIBUTES command, which is described in Section 3.5.6.3.

This command also provides the address and size of the shared memory region. For a given performance domain, and for each performance level or level index in that domain, statistics in the shared memory region track the number of times that the level has been used and the amount of time that the domain has been in that performance level. The statistics must be updated regardless of the agent in the system that placed a domain into a given performance level. After a system reset or shutdown, all the statistics must be initialized to zero when the system first starts up. Time measurements are in microseconds.

For APs, the shared memory must be accessible from the Non-secure world and must be mapped as non-cached normal memory or device memory. The format of the shared memory structure is described in Table 15.

Table 15: Performance level statistics memory region

Field	Byte Length	Byte Offset	Description
Signature	0x4	0x0	0x50455246 ('PERF').
Revision	0x2	0x4	For this revision, this value must be 0x1.
Attributes	0x2	0x6	For this revision, this value must be zero.
Number of domains	0x2	0x8	Number of domains for which statistics are collected.
Reserved	0x2	0xA	Must be zero.
Match Sequence	0x4	0xC	The match sequence populated by the platform as described in Section 3.3.4.1.
Performance domain offset array	0x4 × (Number of domains)	0x10	For each performance domain, this array provides a 4-byte offset, from the start of the shared memory area, to the memory location of the performance domain entry in the data section. The entry format is described in Table 16. A value of zero for the offset of a given performance domain indicates that statistics are not collected for that domain.
Performance domain data section	--	--	This area must start at an offset of 0x10 + 0x4 × (Number of performance domains), or higher.

The performance domain data section contains entries for each performance domain. The format for each entry is described in Table 16.

Table 16: Performance domain entry

Field	Byte Length	Byte Offset	Description
Number of performance levels	0x2	0x0	Number of performance level or level index entries in the performance levels array.
Current performance level index	0x2	0x2	Index into performance level array for current performance level, or level index.
Extended statistics table offset	0x4	0x4	Contains the 4-byte offset, from the start of shared memory, to the start of the domain's Extended Statistics Table. This field is set to 0 if the Extended Statistics table is not supported. The Extended Statistics table definition is implementation specific.
Time of last change	0x8	0x8	Timestamp in microseconds since boot, of the last performance level transition.
Performance level array	N × 0x18	0x10	Performance level array, where N is the number of performance levels or level indices. Refer to Table 17 for more details.

The format for each entry in the performance level array is described in Table 17.

Table 17: Performance level array entry

Field	Byte Length	Byte Offset	Description
Performance level	0x4	0x0	Performance level, or level index.
Reserved	0x4	0x4	Reserved, must be set to zero.
Usage count	0x8	0x8	Number of times this domain has used this performance level. This value must be updated when the domain transitions into the performance level.
Residency	0x8	0x10	This value represents the amount of time domain has been running at the performance level and is given in microseconds. This value must be updated every time the domain transitions to different performance level.

Accessing statistics can cause races between platform write accesses and agent read accesses. This problem and its solution are described in Section 3.3.4.1.

3.6 Clock management protocol

This protocol is intended for the management of clocks. It is used to enable or disable clocks, and to set rates. The protocol provides commands to:

- Describe the protocol version.
- Discover implementation attributes.
- Describe a clock.
- Enable or disable a clock.
- Set the rate and other configuration of the clock synchronously or asynchronously.
- Get and set the parent of a clock.
- Optionally, get the permissions associated with a clock.
- Optionally receive notifications when the rate of a clock is changed or when the platform receives a request for a clock rate change.

3.6.1 Clock management protocol background

This protocol can be used for managing clock rates. It is not to be confused with the performance management protocol, which is used to manage the speed of compute engines such as application processors or GPUs. Examples of usage for the clock protocol might be setting rates for LCD clocks or I^2C buses and registering for notifications issued on clock rate changes.

The protocol does not cover discovery of the clock tree, which must be described through firmware tables instead.

Protocol commands take integer identifiers to describe which clock a given command applies to. The identifiers are sequential and start from 0.

Platform policy dictates which agents can access a clock device, and whether a clock is shared or exclusive. For all combinations of clocks and agents, platform policy dictates which clocks an agent can access. This policy is guided by the required agent functionality while maintaining the security, confidentiality, and integrity of other agents in the system. When a clock is shared among multiple agents, the clock must be in a state that can ensure service guarantees to all agents. For example, a shared clock device needs to be enabled if any of the agents sharing it has requested the clock to be enabled. Similarly, a shared clock can only be disabled if no agent has requested it to be enabled.

The rate that a shared clock is set to is dictated by platform specific policy. While an example policy might be to set it to the highest requested clock rate, it might not be appropriate for all scenarios. As a result, specifying a clock rate management policy for shared clocks is out of scope of this specification, except mandating that the clock must be in a state that can ensure service guarantees to all agents.

The protocol allows an agent to discover the possible parents of a clock device, and to change the parent clock if needed. Possible parents are those clocks which can be selected as an input clock of the clock device. This is done by typically configuring a clock multiplexer or via other platform specific mechanisms.

Figure 5 shows an example where CLK-D can be derived from any one of the parent clocks CLK-A, CLK-B or CLK-C by appropriately configuring the multiplexer MUX.

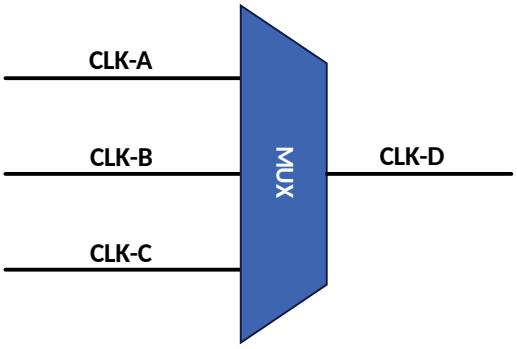

Figure 5: Parent clocks

Note

For security reasons, the clock management protocol should not be used:

- to control the performance of Processing Elements (PEs), whether they are application processors or otherwise.
- if a voltage needs to be adjusted alongside changing the clock rate to ensure correct functioning of the device or peripheral which is supplied by the clock device.

The Performance domain management protocol should be used in all the above cases.

3.6.2 Commands

3.6.2.1 PROTOCOL_VERSION

On success, this command returns the version of this protocol. For this version of the specification, the return value must be `0x30000`, which corresponds to version 3.0.

message_id: `0x0`
protocol_id: `0x14`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this value must be `0x30000`.

3.6.2.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions

different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: `0x10`
protocol_id: `0x14`
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.6.2.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: `0x1`
protocol_id: `0x14`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 attributes	**Bits[31:24]** Reserved, must be zero. **Bits[23:16]** Maximum number of pending asynchronous clock rate changes supported by the platform. **Bits[15:0]** Number of clocks.

3.6.2.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

This command can be used to inquire if clock rate change notifications are supported, by passing CLOCK_RATE_NOTIFY or CLOCK_RATE_CHANGE_REQUESTED_NOTIFY message identifier to the call. If the platform returns SUCCESS, then it supports clock rate change notifications. Otherwise, if the platform returns NOT_FOUND, then it is an indication that notifications are not implemented, or that notifications are not available to the calling agent. The notifications commands are described in Section 3.6.2.12 and Section 3.6.2.13.

message_id: 0x2
protocol_id: 0x14
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is invalid or not provided by this platform implementation. See Section 3.1.4 for more status code definitions.
uint32 attributes	Flags that are associated with a specific command in the protocol. For all commands in this protocol, this parameter has a value of 0.

3.6.2.5 CLOCK_ATTRIBUTES

This command returns the attributes that are associated with a specific clock. An agent might be allowed access to only a subset of the clocks available in the system. The platform must thus guarantee that clocks that an agent cannot access are not visible to it.

Note

Even if this command reports that a clock is enabled, an agent which requires the clock to be enabled must explicitly request for it using the CLOCK_CONFIG_SET command as specified in Section 3.6.2.9. If CLOCK_CONFIG_SET is not called by the agent, the platform can disable the clock at any time due to power saving or other reasons. This rule is not applicable if the clock state cannot be controlled by the agent. CLOCK_GET_PERMISSIONS command, as specified in Section 3.6.2.17, can be used to discover if the clock state can be controlled by the agent.

message_id: 0x3
protocol_id: 0x14
This command is mandatory.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid clock attributes are returned. • NOT_FOUND: if clock_id does not point to a valid clock device. See Section 3.1.4 for more status code definitions.

uint32 attributes	**Bit[31]** Clock rate change notifications support. • Set to 1 if clock rate change notifications are supported for this clock. • Set to 0 if clock rate change notifications are not supported for this clock. **Bit[30]** Clock rate change requested notifications support. • Set to 1 if clock rate change requested notifications are supported for this clock. • Set to 0 if clock rate change requested notifications are not supported for this clock. **Bit[29]** Extended Clock name. • If set to 1, the clock name is greater than 16 bytes. The extended clock name is provided by CLOCK_NAME_GET command which is described in Section 3.6.2.10. • If set to 0, extended clock name is not supported. **Bit[28]** Parent clock identifier support. • Set to 1 if parent clock identifiers are advertised for this clock. • Set to 0 if parent clock identifiers are not advertised for this clock. **Bit[27]** Extended configuration support. • Set to 1 if extended configurations are supported for this clock. Extended configurations can be accessed using the CLOCK_CONFIG_SET and the CLOCK_CONFIG_GET command, as specified by Section 3.6.2.9 and Section 3.6.2.10 respectively. • Set to 0 if extended configurations are not supported for this clock. **Bits[26:2]** Reserved, must be zero. **Bit[1]** Restricted clock. • Set to 1 if the clock has restrictions on changing some of its configuration or settings, and the CLOCK_GET_PERMISSIONS command, as specified in Section 3.6.2.17, can be used to discover the restrictions in place. • Set to 0 if either of the following are true: – The clock's restrictions cannot be discovered because CLOCK_GET_PERMISSIONS is not implemented. – The clock has no restrictions on changing its configuration or setting. Attempts to change a restricted clock configuration or setting returns DENIED. **Bit[0]** Enabled/disabled. • If set to 1, the clock device is enabled. • If set to 0, the clock device is disabled.
uint8 clock_name[16]	A NULL terminated ASCII string with the clock name, of up to 16 bytes. When Bit[29] of attributes field is set to 1, this field contains the lower 15 bytes of the NULL terminated clock name.

uint32 clock_enable_delay	The worst-case delay incurred by the platform to enable the clock in response to a clock enable request from an agent. This field is specified in microseconds. If set to 0, this field is not supported by the platform.

3.6.2.6 CLOCK_DESCRIBE_RATES

This command allows the agent to ascertain the valid rates to which the clock can be set. On success, the command returns an array, which contains several rate entries. Sometimes it might not be possible to return the whole clock rate array with just one call. To solve this problem, the interface allows multiple calls. It also returns the number of remaining clock rates. The size of the array returned depends on the number of return values a given transport can support.

Clocks can support many rates and sometimes individually describing each rate might be too onerous. In such cases, the command can return only the lowest rate, the highest rate, and the step size between two successive physical rates that the clock device can synthesize.

The clock rates returned by this call should be in numeric ascending order.

message_id: `0x4`
protocol_id: `0x14`
This command is mandatory.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.
uint32 rate_index	Index to the first rate value to be described in the return rate array.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid clock rates were returned. • NOT_FOUND: if the clock identified by clock_id does not exist. • OUT_OF_RANGE: if the rate_index is outside of valid range. See Section 3.1.4 for more status code definitions.

uint32 num_rates_flags	Descriptor for the rates supported by this clock. **Bits[31:16]** Number of remaining rates. This field should be 0 if Bit[12] is 1. **Bits[15:13]** Reserved, must be zero. **Bit[12]** Return format: • If this bit is set to 1, the Rate Array is a triplet that constitutes a segment in the following form: – rates[0] is the lowest physical rate that the clock can synthesize in the segment. – rates[1] is the highest physical rate that the clock can synthesize in the segment. – rates[2] is the step size between two successive physical rates that the clock can synthesize within the segment. • If this bit is set to 0, each element of the Rate Array represents a discrete physical rate that the clock can synthesize. **Bits[11:0]** Number of rates that are returned by this call. This field should be 3 if Bit[12] is 1.
{uint32, uint32} rates [N]	Rate Array: If Bit[12] of the num_rates_flags field is set to 0, each array entry is composed of two 32-bit words and has the following format: • Lower word: Lower 32 bits of the physical rate in Hertz. • Upper word: Upper 32 bits of the physical rate in Hertz. If Bit[12] of the num_rates_flags field is set to 1, then each entry is a member of a segment {lowest rate, highest rate, step size} as described above. N is specified by Bits[11:0] of num_rates_flags field.

For an example of using this kind of API, see Section 3.5.6.8.

3.6.2.7 CLOCK_RATE_SET

This command allows the caller to select the clock rate of a clock synchronously or asynchronously.

The command returns when the clock rate has been changed. If a clock is in disabled state, the new rate takes effect when the clock has been re-enabled.

message_id: `0x5`
protocol_id: `0x14`
This command is mandatory.

Parameters

Name	Description

uint32 flags	**Bits[31:4]** Reserved, must be zero. **Bits[3:2]** Round up/down: • If Bit[3] is set to 1, the platform rounds up/down autonomously to choose a physical rate closest to the requested rate, and Bit[2] is ignored. • If Bit[3] is set to 0, the platform: – rounds up if Bit[2] is set to 1. – rounds down if Bit[2] is set to 0. **Bit[1]** Ignore delayed response: • If the Async flag, bit[0], is set to 1 and this bit is set to 1, the platform does not send a CLOCK_RATE_SET delayed response. • If the Async flag, bit[0], is set to 1 and this bit is set to 0, the platform does send a CLOCK_RATE_SET delayed response. • If the Async flag, bit[0], is set to 0, then this bit field is ignored by the platform. **Bit[0]** Async flag: • Set to 1 if clock rate is to be set asynchronously. In this case the call is completed with CLOCK_RATE_SET_COMPLETE message if bit[1] is set to 0. A SUCCESS return code in this case indicates that the platform has successfully queued this command. For more information, see Section 3.6.3.1. • Set 0 to if the clock rate is to be set synchronously. In this case, the call will return when the clock rate setting has been completed.
uint32 clock_id	Identifier for the clock device.
uint32 rate[2]	Lower word: Lower 32 bits of the physical rate in Hertz. Upper word: Upper 32 bits of the physical rate in Hertz.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the clock rate was set successfully for a synchronous request or if the command was successfully enqueued for an asynchronous request. • NOT_FOUND: if the clock identified by clock_id does not exist. • INVALID_PARAMETERS: if the requested rate is not supported by the clock, or the flags parameter specifies invalid or illegal options. • BUSY: if there are too many asynchronous clock rate changes pending. The PROTOCOL_ATTRIBUTES command provides the maximum number of pending asynchronous clock rate changes supported by the platform. • DENIED: if the clock rate cannot be set because of dependencies, e.g., if there are other users of the clock. See Section 3.1.4 for more status code definitions.

3.6.2.8 CLOCK_RATE_GET

This command allows the calling agent to request the current clock rate

If the clock is in disabled state, this command returns the rate at which the clock device would be subsequently running when it has been re-enabled.

Note

If the clock rate is set asynchronously, the rate value that is returned by this command might be stale by the time the command completes.

message_id: `0x6`
protocol_id: `0x14`
This command is mandatory.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the current clock rate was successfully returned. • NOT_FOUND: if the clock identified by clock_id does not exist. See Section 3.1.4 for more status code definitions.
uint32 rate[2]	Lower word: Lower 32 bits of the physical rate in Hertz. Upper word: Upper 32 bits of the physical rate in Hertz.

3.6.2.9 CLOCK_CONFIG_SET

This command allows the calling agent to configure a clock device.

message_id: `0x7`
protocol_id: `0x14`
This command is mandatory.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.

uint32 attributes	**Bits[31:24]** Reserved, must be zero. **Bits[23:16]** Extended config type, as specified in Table 18. A value of 0 indicates that this field is unused. **Bits[15:2]** Reserved, must be zero. **Bits[1:0]** Enable/Disable: • If set to 3, the state of the clock device is unchanged. It is invalid to use this value if Bits[23:16] is set to 0. • The value of 2 is reserved for future use. • If set to 1, the clock device is to be enabled. • If set to 0, the clock device is to be disabled.
uint32 extended_config_val	The specified configuration value corresponding to the extended configuration type specified by Bits[23:16] of attributes field. This field is used to set extended configuration of the clock device. It can be ignored if extended config type specified by Bits[23:16] of the attributes field is set to 0.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the clock configuration has been set successfully. • NOT_FOUND: if the clock identified by clock_id does not exist. • INVALID_PARAMETERS, if the input attributes flag specifies unsupported or invalid configurations. See Section 3.1.4 for more status code definitions.

Table 18: Extended clock config types

Value	Description	Units
0x0	Not used	-
0x1	Duty cycle	Percentage (%)
0x2	Phase	Milliradians (mrad)
0x3-0x7F	Reserved for future use	-
0x80-0xFF	OEM defined	OEM defined

3.6.2.10 CLOCK_CONFIG_GET

This command allows the calling agent to get the configuration of a clock device.

Note

Even if this command reports that a clock is enabled, an agent which requires the clock to be enabled must explicitly request for it using the CLOCK_CONFIG_SET command as specified in Section 3.6.2.9. If CLOCK_CONFIG_SET is not called by the agent, the platform can disable the clock at any time due to power saving or other reasons. This rule is not

applicable if the clock state cannot be controlled by the agent. CLOCK_GET_PERMISSIONS command, as specified in Section 3.6.2.17, can be used to discover if the clock state can be controlled by the agent.

message_id: 0xB
protocol_id: 0x14
This command is mandatory.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.
uint32 flags	**Bits[31:8]** Reserved, must be zero. **Bits[7:0]** Extended config type, as specified in Table 18. A value of 0 indicates that this field is unused.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the clock configuration has been successfully returned. • NOT_FOUND: if the clock identified by clock_id does not exist. • INVALID_PARAMETERS, if the input attributes field of the command specifies unsupported or invalid configurations. See Section 3.1.4 for more status code definitions.
uint32 attributes	Reserved, must be zero.
uint32 config	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Enable/Disable • If set to 1, the clock device is enabled. • If set to 0, the clock device is disabled.
uint32 extended_config_val	The configuration value corresponding to the extended configuration type specified by Bits[7:0] of the attributes field of the command. This field is ignored if the extended config type field specified by Bits[7:0] of the attributes field of the command is set to 0.

3.6.2.11 CLOCK_NAME_GET

This command returns the extended name associated with a specific clock device.

message_id: 0x8
protocol_id: 0x14
This command is mandatory only if extended clock name is supported.
Extended clock name is supported when Bit[29] of attributes field of CLOCK_ATTRIBUTES command is set to 1.
Refer to Section 3.6.2.5 for more details.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid clock extended name is returned. • NOT_FOUND: if clock_id pertains to a non-existent clock. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint8 name[64]	Null-terminated ASCII string of up to 64 bytes in length describing the clock extended name.

3.6.2.12 CLOCK_RATE_NOTIFY

This command allows the caller to request notifications from the platform for clock rate changes on a specific clock device. These notifications are sent using the CLOCK_RATE_CHANGED notification, which is described in Section 3.6.4.1.

Notification support is optional, and PROTOCOL_MESSAGE_ATTRIBUTES must be used to discover whether this command is implemented.

These notifications must be disabled by default during initial boot of the platform.

message_id: `0x9`
protocol_id: `0x14`
This command is optional.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.
uint32 notify_enable	**Bits[31:1]** Reserved must be zero. **Bit[0]** Notify enable. This bit can have one of the following values: • 0, which indicates that the platform does not send any CLOCK_RATE_CHANGED messages to the calling agent. • 1, which indicates that the platform sends CLOCK_RATE_CHANGED messages to the calling agent when the clock rate for the clock device is changed. See Section 3.6.4.1 for details about the CLOCK_RATE_CHANGED notification.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if clock_id does not point to a valid clock device. • INVALID_PARAMETERS: if notify_enable specifies values that are either illegal or incorrect. See Section 3.1.4 for more status code definitions.

3.6.2.13 CLOCK_RATE_CHANGE_REQUESTED_NOTIFY

This command allows the caller to receive notifications when the platform receives a request from another agent or entity in the system, including the platform itself, to change the rate of a clock device. These notifications are sent using the CLOCK_RATE_CHANGE_REQUESTED notification, which is described in Section 3.6.4.2.

CLOCK_RATE_CHANGE_REQUESTED notifications are useful for the co-operative management of clocks. For example, this notification can be used when the platform wants to change the rate of a clock device for thermal management reasons and chooses to co-ordinate with agents using the clock to co-operatively decide whether the clock rate change should proceed.

The platform will notify interested agents of its intent to change the clock rate through the CLOCK_RATE_CHANGE_REQUESTED notification.

This notification enables the agents to either allow or disallow the clock rate transition to proceed. The agents can voluntarily request a change in the rate of the clock device by calling CLOCK_RATE_SET to match the rate specified by the notification, thereby allowing the clock rate transition to proceed.

If the agents choose to ignore this notification and take no further action, the platform will not transition the clocks to the requested rate. However, in exceptional circumstances like thermal runaway, the platform can still proceed to change the clock rate. The exact behavior of the platform and the agents is based on an implementation-defined policy and is out of the scope of this specification.

Notification support is optional, and PROTOCOL_MESSAGE_ATTRIBUTES must be used to discover whether this command is implemented.

These notifications must be disabled by default during initial boot of the platform.

message_id: `0xA`
protocol_id: `0x14`
This command is optional.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.

uint32 notify_enable	**Bits[31:1]** Reserved must be zero. **Bit[0]** Notify enable. This bit can have one of the following values: • 0, which indicates that the platform does not send CLOCK_RATE_CHANGE_REQUESTED messages to the calling agent. • 1, which indicates that the platform sends CLOCK_RATE_CHANGE_REQUESTED messages to the calling agent when another agent or entity in the system, including the platform itself, requests for a change in the clock rate. See Section 3.6.4.2 for more details about the CLOCK_RATE_CHANGE_REQUESTED notification.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if clock_id does not point to a valid clock device. See Section 3.1.4 for more status code definitions.

3.6.2.14 CLOCK_POSSIBLE_PARENTS_GET

This command allows the calling agent to get all the possible parents of a clock device. However, there can be only one parent actively associated with a clock device at any point in time.

This command is useful in constructing a clock topology when changing the parent clock of a clock device is sometimes required due to implementation defined considerations like jitter, power, or other factors. Section 3.6.2.15 specifies the command to change the parent of a clock device.

Sometimes it might not be possible to return the complete list of all possible parents in one call due to transport size limitations. To solve this problem, the interface allows multiple calls.

The call also provides the number of parents which it was unable to return. The size of the array returned depends on the number of return values that a given transport can support.

The parents returned by this call should be in numeric ascending order of their identifiers.

message_id: `0xC`
protocol_id: `0x14`
This command is mandatory if for any clock device, Bit[28] of the attributes field as returned by the CLOCK_ATTRIBUTES command is set to 1.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.
uint32 skip_parents	The number of parents to skip over, before returning the first possible parent in the return parent array.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if the possible clock parents have been successfully returned. • NOT_FOUND: if the clock identified by clock_id does not exist. • OUT_OF_RANGE: if the skip_parents parameter is outside of the valid range. • NOT_SUPPORTED: if advertising the parents for the clock specified by clock_id is not supported. • DENIED: if the calling agent is not allowed to get the possible parents. See Section 3.1.4 for more status code definitions.
uint32 flags	Descriptor for the parents supported by this clock device. **Bits[31:24]** Number of remaining parents. **Bits[23:8]** Reserved, must be zero. **Bits[7:0]** Number of parent clock identifiers that are returned by this call.
uint32 possible parents[N]	Array of possible parent clocks listed in numeric ascending order of their identifiers. N is specified by Bits[7:0] of num_parents_flags field.

3.6.2.15 CLOCK_PARENT_SET

This command allows the calling agent to change the parent of a clock device. This command does not result in a change of the configured clock rate, or the range of clock rates supported by its children clock devices.

However, in some cases changing the parent of a clock device might not be possible without implicitly affecting other children clock devices sharing the same parent. In these cases, it is recommended that all children clock devices affected by this change should be explicitly disabled by the agent and configured to the expected properties that the new parent supports, before sending this command.

message_id: `0xD`
protocol_id: `0x14`
This command is mandatory if for any clock device, Bit[28] of the attributes field as returned by the CLOCK_ATTRIBUTES command is set to 1.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.
uint32 parent_id	Identifier for the clock device to set as the parent of the clock specified by clock_id.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if the clock parent has been set successfully. • NOT_FOUND: if the clock identified by clock_id or clock_parent_id does not exist. • INVALID_PARAMETERS: if the parent clock identifier specified is not a valid parent of the clock specified by clock_id. • OUT_OF_RANGE: changing the parent clock is not possible because of inability to maintain child clock requirements. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to set the parent. See Section 3.1.4 for more status code definitions.

3.6.2.16 CLOCK_PARENT_GET

This command allows the calling agent to get the current parent of a clock device.

message_id: `0xE`
protocol_id: `0x14`
This command is mandatory if for any clock device, Bit[28] of the attributes field as returned by the CLOCK_ATTRIBUTES command is set to 1.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the clock parent has been successfully returned. • NOT_FOUND: if the clock identified by clock_id does not exist. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to get the parent. See Section 3.1.4 for more status code definitions.
uint32 parent_id	Identifier for the parent clock device.

3.6.2.17 CLOCK_GET_PERMISSIONS

An agent might be restricted from changing certain configuration or settings of a clock. This command returns the restrictions that are associated with a specific clock.

message_id: `0xF`
protocol_id: `0x14`
This command is mandatory if Bit[1] of the attributes field returned by the CLOCK_ATTRIBUTES command, as specified in Section 3.6.2.5, is set to 1 for any clock. This command is optional in all other cases.

Parameters

Name	Description
uint32 clock_id	Identifier for the clock device.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid clock permissions are returned. • NOT_FOUND: if clock_id does not point to a valid clock device. • NOT_SUPPORTED: if the request is not supported. See Section 3.1.4 for more status code definitions.
uint32 permissions	**Bit[31]** Clock state control • Set to 1 if the clock can be disabled or enabled by the agent. • Set to 0 if the clock state cannot be changed by the agent. Attempts to change the clock state using CLOCK_CONFIG_SET command returns DENIED. **Bit[30]** Clock parent control • Set to 1 if the clock parent can be changed by the agent. • Set to 0 if the clock parent cannot be changed by the agent. CLOCK_PARENT_SET command returns DENIED. **Bit[29]** Clock rate control • Set to 1 if the clock rate can be changed by the agent. • Set to 0 if the clock rate cannot be changed by the agent. CLOCK_RATE_SET command returns DENIED. **Bits[28:0]** Reserved, must be zero.

3.6.3 Delayed responses

3.6.3.1 CLOCK_RATE_SET_COMPLETE

If the agent has changed the clock rate asynchronously through CLOCK_RATE_SET, the platform sends this delayed response to the agent when the clock rate changes.

message_id: `0x5`
protocol_id: `0x14`
This command is optional.

Parameters

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if clock rate was set successfully. • DENIED: if the request was denied because there are other users of the clock. Other vendor-specific errors can also be generated depending on the implementation. See Section 3.1.4 for more status code definitions.
uint32 clock_id	Identifier for the clock device.

uint32 rate[2]	Value of the rate that the clock transitioned to. Lower word: Lower 32 bits of the physical rate in Hertz. Upper word: Upper 32 bits of the physical rate in Hertz.

3.6.4 Notifications

3.6.4.1 CLOCK_RATE_CHANGED

If an agent has registered to receive clock rate change notifications for the clock device that is identified by clock_id, the platform sends these notifications to that agent when the clock rate changes.

The platform sends this notification to a subscribing agent when the clock rate of the clock device is changed by a different agent or entity in the system, including the platform itself. The platform might autonomously change the clock rate of the clock device to apply thermal or other constraints. In each of these occurrences, the subscribing agent will be notified so that it can become aware of the change. These notifications are sent after the clock rate transition has completed.

message_id: `0x0`
protocol_id: `0x14`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent that caused the clock rate change.
uint32 clock_id	Identifier of the clock device.
uint32 rate[2]	Value of the rate that the clock has transitioned to. Lower word: Lower 32 bits of the physical rate in Hertz. Upper word: Upper 32 bits of the physical rate in Hertz.

3.6.4.2 CLOCK_RATE_CHANGE_REQUESTED

An agent might have registered, via CLOCK_RATE_CHANGE_REQUESTED_NOTIFY, to receive notifications when there is a request from a different agent or entity in the system, including the platform itself, to change the rate of a clock device. The platform sends this notification to the subscribing agent when such a request is received by it.

For more details on how CLOCK_RATE_CHANGE_REQUESTED notifications can be used, see Section 3.6.2.13.

message_id: `0x1`
protocol_id: `0x14`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent that requested the clock rate change.
uint32 domain_id	Identifier of the clock device.
uint32 rate[2]	The clock rate requested. Lower word: Lower 32 bits of the physical rate in Hertz. Upper word: Upper 32 bits of the physical rate in Hertz.

3.7 Sensor management protocol

This protocol provides functions to manage platform sensors, and provides the following commands:

- Describe the protocol version.
- Describe the attribute flags of the protocol.
- Discover sensors that are implemented and managed by the platform.
- Read a sensor synchronously or asynchronously as allowed by the platform.
- Obtain and program sensor attributes, if applicable.
- Configure a sensor.
- Receive notifications on specific changes to sensor data, for example when a sensor value crosses a threshold.
- Receive continuous sensor updates through notifications.
- Specify a region of shared memory for conveying sensor values, if supported by the platform.

3.7.1 Sensor management protocol background

This protocol supports sensors which measure and report values along one or more axes like a 3-axis accelerometer. It also supports sensors which measure a scalar value like temperature. The protocol allows each axis of a multi-axis sensor to specify its own unit of measurement and other attributes. A sensor reporting uncalibrated values can report bias estimates as additional axes of measurement.

Protocol commands take integer identifiers to identify the sensor they apply to. The identifiers are sequential and start from 0.

The protocol supports accessing sensors through one of the following mechanisms:

- **Synchronous Access** – This method is recommended for sensors whose data is immediately available or is internally cached by the platform and can be returned immediately to the requesting agent. Examples include platform event counters, or sensor data samples that are stored in internal memory within the platform.
- **Asynchronous Access** – This method is recommended for sensors whose data is not cached by the platform or for sensors that are slow to read. An example of this could be an on-die thermal sensor.
- **Event Notification** – The agent can register for receiving notifications on specific sensor values, conditions, or states of interest. The agent can also register for receiving sensor values at a configured minimum update interval, if supported by the sensor.
- **Shared Memory** – In this scheme, the platform periodically updates the sensor value in an area of memory that is shared between agents and the platform.

Agents can discover the access mechanisms that are supported by a particular sensor by examining the attributes and descriptors that are advertised for the sensor. The platform can support multiple access mechanisms.

For all combinations of sensors and agents, platform policy dictates which sensors an agents can access. This policy is guided by the required agent functionality while maintaining the security, confidentiality, and integrity of other agents in the system.

—— **Note** ——————————

Sensors can be exploited for side-channel attacks because they can be used to accurately observe system behavior when sensitive workloads are executing. It is recommended that this protocol is enabled only after due consideration of the potential possibility and mechanism of such an attack. It is also recommended that the sensor measurement granularity and update interval is restricted on production systems after considering the worst-case data dependent variation of the system properties which the sensor is measuring. Additional defense mechanisms might include, but are not restricted to, the introduction of random noise margins to measurements and certificate-based enablement of relevant platform

functionality. It is the responsibility of the platform to ensure that this protocol is not exploitable for attacks. The exact mechanism used to identify the deployment parameters of this protocol is beyond the scope of this specification.

3.7.2 Commands

3.7.2.1 PROTOCOL_VERSION

On success, this command returns the version of this protocol. For this version of the specification, the return value must be `0x30001`, which corresponds to version 3.1.

message_id: `0x0`
protocol_id: `0x15`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this value must be `0x30001`.

3.7.2.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: `0x10`
protocol_id: `0x15`
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.7.2.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: 0x1
protocol_id: 0x15
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 attributes	**Bits[31:24]** Reserved, must be zero. **Bits[23:16]** Maximum number of outstanding asynchronous commands that are supported by the platform. **Bits[15:0]** Number of sensors that are present.
uint32 sensor_reg_address_low	This value indicates the lower 32 bits of the physical address where the sensor shared memory region is located. This value should be 64-bit aligned. The address must be in the memory map of the calling agent. If the sensor_reg_len field is 0, then this field is invalid and must be ignored by the agent.
uint32 sensor_reg_address_high	This value indicates the upper 32 bits of the physical address where the shared memory region is located. The address must be in the memory map of the calling agent. If the sensor_reg_len field is 0, then this field is invalid and must be ignored by the agent.
uint32 sensor_reg_len	This value indicates the length in bytes of the shared memory region. A value of 0 in this field indicates that the platform does not implement the sensor shared memory.

Note: Statistics capability has been marked for deprecation and is being replaced by system telemetry.

The sensor shared memory region is described in Section 3.7.5.

3.7.2.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

If the message is not supported or implemented by the platform, then this command returns a NOT_FOUND status code. This allows calling agents to comprehend which commands are supported on a platform and configure themselves accordingly.

message_id: 0x2
protocol_id: 0x15
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is not provided by this platform implementation. Other status codes according to Section 3.1.4 might be returned for general error or status reporting.
uint32 attributes	Attributes that are associated with the message that is specified by message_id. Currently, this field returns the value of 0.

3.7.2.5 SENSOR_DESCRIPTION_GET

This command can be used for sensor discovery on the platform. On success, it returns an array of Sensor Descriptors as described in Sensor Descriptor.

message_id: `0x3`
protocol_id: `0x15`
This command is mandatory.

Parameters

Name	Description
uint32 desc_index	Index of the first sensor descriptor to be read in the sensor descriptor array.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 num_sensor_flags	**Bits[31:16]** Number of remaining sensor descriptors. **Bits[15:12]** Reserved, must be zero. **Bits[11:0]** Number of sensor descriptors that are returned by this current call.
SENSOR_DESC desc[N]	An array of sensor descriptors of format described in Sensor Descriptor.

Sensor Descriptor

The SENSOR_DESC structure describes the sensor properties, such as the unique identifier for the sensor, its name, number of axes, reading types and other characteristics.

uint32 sensor_id	Identifier for the sensor.

uint32 sensor_attributes_low	**Bit[31]** Asynchronous sensor read support: • If this flag is set to 1, then this sensor can be read asynchronously through the SENSOR_READING_GET command, and its value is returned in the SENSOR_READING_COMPLETE delayed response. • If this flag is set to 0, the sensor must only be read using a synchronous call to SENSOR_READING_GET command. **Bit[30]** Continuous sensor update notification support: • If this flag is set to 1, the sensor supports continuous update notifications. The platform sends the SENSOR_UPDATE notification when an updated sensor value is available. The platform sends this notification only if the sensor is enabled and the agent has subscribed to the notification by calling SENSOR_CONTINUOUS_UPDATE_NOTIFY command. • If this flag is set to 0, the sensor does not support continuous update notifications. The SENSOR_CONTINUOUS_UPDATE_NOTIFY command is not available for this sensor. The sensor values need to be read using the SENSOR_READING_GET command. **Bit[29]** Extended sensor name. • If set to 1, the sensor name is greater than 16 bytes. The extended sensor name is provided by SENSOR_NAME_GET command which is described in Section 3.7.2.14. • If set to 0, extended sensor name is not supported. **Bits[28:15]** Reserved for future use. **Bit[14:10]** Timestamp exponent: This field is represented in two's complement format. It is the power-of-10 multiplier that is applied to the sensor timestamps (timestamp x $10^{[\text{timestamp exponent}]}$) to represent it in seconds. This field is only valid if Bit[9] is set to 1. **Bit[9]** Timestamp support: • If this flag is set to 1, the sensor can provide timestamped values. • If this flag is set to 0, the sensor cannot provide timestamped values. Timestamps should be derived from a monotonic time base. The selection of a time base is beyond the scope of this specification and should be agreed between the agent and the platform by other standard mechanisms. **Bit[8]** Extended attributes support: • If this flag is set to 1, the sensor reports extended attributes after the sensor_name field. • If this flag is set to 0, the sensor does not report extended attributes. Fields beyond sensor_name are not allocated by the platform. **Bits[7:0]** Number of trip points supported.

uint32 sensor_attributes_high	**Bits[31:22]** Reserved for future use. **Bits[21:16]** Number of axes: The number of axes of measurement the sensor supports. • This field is valid only if Bit[8] is set to 1. • This field must not be 0 if Bit[8] is set to 1. **Bits[15:11]** Exponent: The power-of-10 multiplier in two's- complement format that is applied to the sensor unit specified by the SensorType field. This field is valid only if Bit[8] is set to 0. To obtain this field for sensors which report values along axes refer to Section 3.7.2.6. **Bits[10:9]** Reserved. **Bit[8]** Axis support: • If this flag is set to 1, the sensor reports values along one or more axes. • If this flag is set to 0, the sensor reports a scalar value like temperature. **Bits[7:0]** SensorType: The type of sensor and the measurement system it implements, as described in Table 19. This field is valid only if Bit[8] is set to 0. To obtain this field for sensors which report values along axes refer to Section 3.7.2.6.
uint8 sensor_name[16]	A NULL terminated UTF-8 format string with the sensor name, of up to 16 bytes. When Bit[29] of sensor_attributes_low field is set to 1, this field contains the lower 15 bytes of the NULL terminated sensor name.
uint32 sensor_power	This is an extended attribute field. It reports the average power consumed by the sensor in microwatts (uW) when it is active. Vendors are not obliged to report sensor power or be accurate in reporting it. If the power consumed is not reported, this field must be set to 0. This field is present only if Bit[8] of sensor_attributes_low is set to 1. This field reports the specification of the sensor and must not be used to return the actual measured power consumption of the sensor at runtime. Repeated calls to SENSOR_DESCRIPTION_GET for the same sensor must return the same values.
uint32 sensor_resolution	**Bits[31:27]** Exponent: The power-of-10 multiplier in two's- complementformat that is applied to the Res field. **Bits[26:0]** Res: The resolution of the sensor. This is an extended attribute field. It reports the resolution of the sensor. The representation is in [res] x $10^{[exponent]}$ format, in units specified by SensorType. SensorType is reported by Bits[7:0] of sensor_attributes_high field. If the sensor does not report its resolution, this field must be set to `0x0`. This field is present only if: • **Bit[8]** of sensor_attributes_high is set to 0 which indicates that the sensor reports scalar values, and • **Bit[8]** of sensor_attributes_low is set to 1. To obtain this field for sensors which report values along axes refer to Section 3.7.2.6.

int32 sensor_min_range_low	This is an extended attribute field. It reports the lower 32 bits of the minimum sensor value that can be measured by the sensor. If the sensor does not report its minimum range, this field must be set to `0x0`. This field is present only if: • **Bit[8]** of sensor_attributes_high is set to 0 which indicates that the sensor reports scalar values, and • **Bit[8]** of sensor_attributes_low is set to 1. To obtain this field for sensors which report values along axes refer to Section 3.7.2.6.
int32 sensor_min_range_high	This is an extended attribute field. It reports the higher 32 bits of the minimum sensor value that can be measured by the sensor. If the sensor does not report its minimum range, this field must be set to `0x80000000`. This field is present only if: • **Bit[8]** of sensor_attributes_high is set to 0 which indicates that the sensor reports scalar values, and • **Bit[8]** of sensor_attributes_low is set to 1. To obtain this field for sensors which report values along axes refer to Section 3.7.2.6.
int32 sensor_max_range_low	This is an extended attribute field. It reports the lower 32 bits of the maximum sensor value that can be measured by the sensor. If the sensor does not report its maximum range, this field must be set to `0xFFFFFFFF`. This field is present only if: • **Bit[8]** of sensor_attributes_high is set to 0 which indicates that the sensor reports scalar values, and • **Bit[8]** of sensor_attributes_low is set to 1. To obtain this field for sensors which report values along axes refer to Section 3.7.2.6.
int32 sensor_max_range_high	This is an extended attribute field. It reports the higher 32 bits of the maximum sensor value that can be measured by the sensor. If the sensor does not report its maximum range, this field must be set to `0x7FFFFFFF`. This field is present only if: • **Bit[8]** of sensor_attributes_high is set to 0 which indicates that the sensor reports scalar values, and • **Bit[8]** of sensor_attributes_low is set to 1. To obtain this field for sensors which report values along axes refer to Section 3.7.2.6.

Table 19: Sensor Type Enumerations[1:]

Enum	Sensor Unit Description	Enum	Sensor Unit Description	Enum	Sensor Unit Description
0	None	**33**	Cubic Meters	**66**	Pascals
1	Unspecified	**34**	Liters	**67**	Counts
2	Degrees C	**35**	Fluid Ounces	**68**	Grams

Enum	Sensor Unit Description	Enum	Sensor Unit Description	Enum	Sensor Unit Description
3	Degrees F	**36**	Radians	**69**	Newton-meters
4	Degrees K	**37**	Steradians	**70**	Hits
5	Volts	**38**	Revolutions	**71**	Misses
6	Amps	**39**	Cycles	**72**	Retries
7	Watts	**40**	Gravities	**73**	Overruns/Overflows
8	Joules	**41**	Ounces	**74**	Underruns
9	Coulombs	**42**	Pounds	**75**	Collisions
10	VA	**43**	Foot-Pounds	**76**	Packets
11	Nits	**44**	Ounce-Inches	**77**	Messages
12	Lumens	**45**	Gauss	**78**	Characters
13	Lux	**46**	Gilberts	**79**	Errors
14	Candelas	**47**	Henries	**80**	Corrected Errors
15	kPa	**48**	Farads	**81**	Uncorrectable Errors
16	PSI	**49**	Ohms	**82**	Square Mils
17	Newtons	**50**	Siemens	**83**	Square Inches
18	CFM	**51**	Moles	**84**	Square Feet
19	RPM	**52**	Becquerels	**85**	Square Centimeters
20	Hertz	**53**	PPM (parts/million)	**86**	Square Meters
21	Seconds	**54**	Decibels	**87**	Radians per second
22	Minutes	**55**	DbA	**88**	Beats per Minute
23	Hours	**56**	DbC	**89**	Meters per second squared
24	Days	**57**	Grays	**90**	Meters per second
25	Weeks	**58**	Sieverts	**91**	Cubic meter per second
26	Mils	**59**	Color Temperature Degrees K	**92**	Millimeters of Mercury
27	Inches	**60**	Bits	**93**	Radians per second squared
28	Feet	**61**	Bytes	**94**	State (binary): 1: enabled/on 0: disabled/off
29	Cubic Inches	**62**	Words (data)	**95**	Bytes per second
30	Cubic Feet	**63**	Doublewords	**-**	All others – reserved
31	Meters	**64**	Quadwords		
32	Cubic Centimeters	**65**	Percentage	**255**	OEM Unit

[1:] This table is derived from the sensorUnits enumeration Table of Distributed Management Task Force (DMTF) specification number DSP 0248 (Platform Level Data Model for Platform Monitoring and Control Specification). It is however not an exact replica of the sensorUnits enumeration Table.

3.7.2.6 SENSOR_AXIS_DESCRIPTION_GET

This command is used for the discovery of the sensor axis properties. On success, it returns an array of Sensor Axis Descriptors as described in Sensor Axis Descriptor.

Sensor axis descriptors should be reported in the normative order of axes. For example, a triaxial accelerometer should return its axis descriptors ordered as x, y, and z.

message_id: `0x7`
protocol_id: `0x15`
This command is mandatory for sensors providing measurements along axis only.

Parameters

Name	Description
uint32 sensor_id	Identifier for the sensor.
uint32 axis_desc_index	Index of the first sensor axis descriptor to be read in the sensor axis descriptor array.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid sensor axis descriptors were returned. • NOT_FOUND: if sensor_id does not point to a valid sensor. • NOT_SUPPORTED: if the sensor does not report values along axis. See Section 3.1.4 for more status code definitions.
uint32 num_axis_flags	**Bits[31:26]** Number of remaining sensor axis descriptors. **Bits[25:6]** Reserved, must be zero. **Bits[5:0]** Number of sensor axis descriptors that are returned by this current call.
SENSOR_AXIS_DESC desc[N]	An array of sensor axis descriptors of format described in Sensor Axis Descriptor. N is specified by Bits[5:0] of num_axis_flags field.

Sensor Axis Descriptor

The SENSOR_AXIS_DESC structure describes the sensor axis properties, such as the axis identifier for the sensor, its name, reading types and other characteristics.

uint32 axis_id	Identifier for the axis of the sensor.

uint32 axis_attributes_low	**Bits[31:8]** Reserved. **Bit[9]** Extended axis name. • If set to 1, the axis name is greater than 16 bytes. The extended axis name is provided by SENSOR_AXIS_NAME_GET command which is described in Section 3.7.2.15. • If set to 0, extended axis name is not supported. **Bit[8]** Extended attributes support: • If this flag is set to 1, the sensor reports extended attributes for this axis after the name field. • If this flag is set to 0, the sensor does not report extended attributes for this axis. Fields beyond name are not allocated by the platform. **Bits[7:0]** Reserved.
uint32 axis_attributes_high	**Bits[31:16]** Reserved. **Bits[15:11]** Exponent: The power-of-10 multiplier in two's-complement format that is applied to the sensor unit specified by the SensorType field. **Bits[10:8]** Reserved. **Bits[7:0]** SensorType: The type of sensor axis and the measurement system it implements, as described in Table 19.
uint8 name[16]	A NULL terminated UTF-8 format string with the sensor axis name, of up to 16 bytes. When Bit[9] of axis_attributes_low field is set to 1, this field contains the lower 15 bytes of the NULL terminated axis name. It is recommended that the name ends with '_' followed by the axis of the sensor in uppercase. For example, the name for the x-axis of a triaxial accelerometer could be "acc_X" or "_X".
uint32 axis_resolution	**Bits[31:27]** Exponent: The power-of-10 multiplier in two's-complement format that is applied to the Res field. **Bits[26:0]** Res: The resolution of the sensor axis. This is an extended attribute field. It reports the resolution of the sensor axis. The representation is in [res] x $10^{[exponent]}$ format, in units specified by SensorType. SensorType is reported by Bits[7:0] of axis_attributes_high field. If the sensor does not report the resolution for this axis, this field must be set to 0x0. This field is present only if Bit[8] of axis_attributes_low is set to 1.

int32 axis_min_range_low	Lower 32 bits of the minimum value that can be measured by this axis. If the sensor does not report the minimum range for this axis, this field must be set to `0x0`. This field is present only if Bit[8] of axis_attributes_low is set to 1.
int32 axis_min_range_high	Higher 32 bits of the minimum value that can be measured by this axis. If the sensor does not report the minimum range for this axis, this field must be set to `0x80000000`. This field is present only if Bit[8] of axis_attributes_low is set to 1.
int32 axis_max_range_low	Lower 32 bits of the maximum value that can be measured by this axis. If the sensor does not report the maximum range for this axis, this field must be set to `0xFFFFFFFF`. This field is present only if Bit[8] of axis_attributes_low is set to 1.
int32 axis_max_range_high	Higher 32 bits of the maximum value that can be measured by this axis. If the sensor does not report the maximum range for this axis, this field must be set to `0x7FFFFFFF`. This field is present only if Bit[8] of axis_attributes_low is set to 1.

3.7.2.7 SENSOR_LIST_UPDATE_INTERVALS

This command allows the agent to ascertain the update intervals supported by the sensor. On success, the command returns an array, which contains a number of update interval entries. Sometimes it might not be possible to return all the sensor update intervals with just one call. To solve this problem, the interface allows multiple calls. It also returns the number of remaining update intervals. The size of the array returned depends on the number of return values a given transport can support.

Sensors can support many update intervals and sometimes individually describing each update interval might be too onerous. In such cases the command can return only the lowest update interval, the highest update interval, and the step size between two successive update intervals that the sensor supports.

The sensor update intervals returned by this call should be in numeric ascending order.

message_id: `0x8`
protocol_id: `0x15`
This command is optional.

Parameters

Name	Description
uint32 sensor_id	Identifier for the sensor.
uint32 index	Index to the first update interval value to be described in the return interval array.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if sensor update intervals were returned successfully. • NOT_FOUND: if the sensor identified by sensor_id does not exist. • OUT_OF_RANGE: if the index is outside of valid range. • NOT_SUPPORTED: if the sensor does not support update intervals. See Section 3.1.4 for more status code definitions.
uint32 flags	Descriptor for the update intervals supported by this sensor. **Bits[31:16]** Number of remaining update intervals. This field should be 0 if Bit[12] is 1. **Bits[15:13]** Reserved, must be zero. **Bit[12]** Return format: If this bit is set to 1, the Interval Array is a triplet that constitutes a segment in the following form: • interval[0] is the lowest update interval that the sensor can support in the segment. • interval[1] is the highest update interval that the sensor can support in the segment. • interval[2] is the step size between two successive update intervals that the sensor can support within the segment. If this bit is set to 0, each element of the Interval Array represents a discrete sensor update interval. **Bits[11:0]** Number of update intervals that are returned by this call. This field should be 3 if Bit[12] is 1.
uint32 intervals[N]	Interval Array: Each array entry has the following format: **Bits[31:21]** Reserved. **Bits[20:5]** sec – Seconds. **Bits[4:0]** exponent – two's complement format representing the power-of-10 multiplier that is applied to the sec field. The representation is in [sec] x $10^{[exponent]}$ format, in units of seconds. If Bit[12] of the flags field is set to 0, each array entry is a discrete sensor update interval. If Bit[12] of the flags field is set to 1, then each entry is a member of a segment {lowest update interval, highest update interval, step size}. N is specified by Bits[11:0] of flags field.

For an example of using this kind of API, see Section 3.5.6.8.

3.7.2.8 SENSOR_TRIP_POINT_NOTIFY

This command is used by the agent to globally control generation of notifications on cross-over events for the trip-points that have been configured using the SENSOR_TRIP_POINT_CONFIG command.

message_id: 0x4
protocol_id: 0x15
This command is optional.

Parameters

Name	Description
uint32 sensor_id	Identifier for the sensor.
uint32 sensor_event_control	**Bits[31:1]** Reserved. **Bit[0]** Globally controls generation of notifications on crossing of configured trip-points pertaining to the specified sensor. • If this bit is set to 1, notifications are sent whenever the sensor value crosses any of the trip-points that have been configured using the SENSOR_TRIP_POINT_CONFIG command. • If this bit is set to 0, no notifications are sent for any of the trip-points.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if sensor_id does not point to an existing sensor. • INVALID_PARAMETERS: if the input sensor_event_control flag contains invalid or illegal settings. • NOT_SUPPORTED: if the platform does not support trip point event notifications for the sensor. See Section 3.1.4 for more status code definitions.

3.7.2.9 SENSOR_TRIP_POINT_CONFIG

This command is used for selecting and configuring a trip-point of interest. Following the successful completion of this command, the platform generates the SENSOR_TRIP_POINT_EVENT event whenever the sensor value crosses the programmed trip point value, provided notifications have been enabled for trip-points globally using the SENSOR_TRIP_POINT_NOTIFY command.

Sensors which report values along multiple axes can optionally support trip points only if all the axes report the same sensor units. In such a case, the SENSOR_TRIP_POINT_EVENT event is generated when any axis crosses the configured threshold.

An agent can use this command for various use-cases. For example:

- The agent can invoke this command twice to program the upper and lower values of a hysteresis band, respectively.
- For a counter-type sensor that is required to fire a notification on reaching a certain count, the agent can issue this command to program the count value.

message_id: 0x5
protocol_id: 0x15
This command is mandatory if at least one of the implemented sensors in the platform supports trip points.

Parameters	
Name	**Description**
uint32 sensor_id	Identifier for the sensor.
uint32 trip_point_ev_ctrl	**Bits[31:12]** Reserved. **Bits[11:4]** trip_point_id: Identifier for the selected trip point. This value should be equal to or less than the total number of trip points that are supported by this sensor as advertised in its descriptor. **Bits[3:2]** Reserved for future use. **Bits[1:0]** Event control for the trip-point: • If set to 0, disables event generation for this trip-point (this is the default state). • If set to 1, enables event generation when this trip-point value is reached or crossed in a positive direction. • If set to 2, enables event generation when this trip-point value is reached or crossed in a negative direction. • If set to 3, enables event generation when this trip-point value is reached or crossed in either direction.
uint32 trip_point_val_low	Lower 32 bits of the sensor value corresponding to this trip-point. The default value is 0.
uint32 trip_point_val_high	Higher 32 bits of the sensor value corresponding to this trip-point. The default value is 0.
Return values	
Name	**Description**
int32 status	One of, but not limited to, the following: • SUCCESS: if the sensor trip point was set successfully. • NOT_FOUND: if sensor_id does not point to an existing sensor. • INVALID_PARAMETERS: if the input parameters specify incorrect or illegal values. • NOT_SUPPORTED: if the platform does not support trip point event notifications for the sensor. See Section 3.1.4 for more status code definitions.

3.7.2.10 SENSOR_CONFIG_GET

This command is used to read the sensor configuration. It returns the configured sensor update interval, the sensor state and if timestamping is enabled.

message_id: `0x9`
protocol_id: `0x15`
This command is mandatory.

Parameters	
Name	**Description**

uint32 sensor_id	Identifier for the sensor.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the sensor configuration was returned successfully. • NOT_FOUND: if sensor_id does not point to an existing sensor. See Section 3.1.4 for more status code definitions.
uint32 sensor_config	**Bits[31:11]** sensor_update_interval: **Bits[31:16]** sec – Seconds. **Bits[15:11]** exponent – two's complement format representing the power-of-10 multiplier that is applied to the sec field. The time duration between successive updates of sensor value. The representation is in the [sec] x $10^{[exponent]}$ format, in units of seconds. This field is set to 0 if the sensor does not require or support an update interval. **Bits[10:2]** Reserved. **Bit[1]** Timestamp reporting: • Set to 1 if the sensor value provided by the platform is timestamped. • Set to 0 if the sensor value provided by the platform is not timestamped. **Bit[0]** Sensor State: • Set to 1 if the sensor is enabled. • Set to 0 if the sensor is disabled.

3.7.2.11 SENSOR_CONFIG_SET

This command is used to configure the sensor update interval and to enable the timestamping of sensor values. This command can also be used to enable or disable the sensor.

If the sensor has been enabled, sensor values can be read using the SENSOR_READING_GET command or notified by the platform through the SENSOR_UPDATE notification. The platform generates the SENSOR_UPDATE notification only if the agent subscribes to this notification by calling SENSOR_CONTINUOUS_UPDATE_NOTIFY as described in Section 3.7.2.13. The platform generates this notification only for sensors which support continuous update notifications.

message_id: `0xA`
protocol_id: `0x15`
This command is mandatory.

Parameters

Name	Description
uint32 sensor_id	Identifier for the sensor.

uint32 sensor_config	**Bits[31:11]** sensor_update_interval: **Bits[31:16]** sec – Seconds. **Bits[15:11]** exponent – two's complement format representing the power-of-10 multiplier that is applied to the sec field. The time duration between successive updates of the sensor value. The representation is in the [sec] x $10^{[exponent]}$ format, in units of seconds. This field should be set to 0 if the sensor update interval does not need to be updated or if the sensor does not support configuring the sensor update interval. **Bits[10:9]** Round up/down: • If Bit[10] is set to 1, the platform rounds up/down autonomously to choose a sensor update interval closest to the requested update interval, and Bit[9] is ignored. • If Bit[10] is set to 0, then the platform rounds up if Bit[9] is set to 1, and rounds down if Bit[9] is set to 0. **Bits[8:2]** Reserved. **Bit[1]** Timestamp reporting: • Set to 1 if the sensor value provided by the platform should be timestamped. • Set to 0 if the sensor value provided by the platform should not be timestamped. If the sensor does not support timestamp reporting or its configuration, this bit should be set to 0. **Bit[0]** Sensor State: • Set to 1 if the sensor should be enabled. • Set to 0 if the sensor should be disabled.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the sensor configuration was set successfully. • NOT_FOUND: if sensor_id does not point to an existing sensor. • INVALID_PARAMETERS: if the input parameters specify incorrect or illegal values. • NOT_SUPPORTED: if the configuration requested by this command is not supported by the sensor. See Section 3.1.4 for more status code definitions.

3.7.2.12 SENSOR_READING_GET

This command requests the platform to provide the current value of the sensor that is represented by sensor_id. The sensor should be in enabled state before using this command. For synchronous mode of access, the platform provides the sensor reading in the response to this command itself. For asynchronous accesses, the platform returns the sensor value in the SENSOR_READING_COMPLETE delayed response.

When the platform notices failure or fault conditions in the sensor or its associated logic or circuitry, it returns the HARDWARE_ERROR status. Other errors pertain to the interface itself and are enumerated in Section 3.1.4.

Agents should assess the sensor attributes to determine the optimal mode of access for the sensor. A slow sensor like

a temperature sensor can be more optimally read asynchronously, while a shared memory-based sensor can be read synchronously.

message_id: `0x6`
protocol_id: `0x15`
This command is mandatory.

Parameters

Name	Description
uint32 sensor_id	The identifier for the sensor to be read.
uint32 flags	**Bits[31:1]** Reserved. **Bit[0]** Async flag: • Set to 1 if the sensor is to be read asynchronously. • Set to 0 to if the sensor is to be read synchronously.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the reading was successfully returned for a synchronous request or if the command was successfully enqueued for an asynchronous request. • NOT_FOUND: if sensor_id does not point to an existing sensor. • INVALID_PARAMETERS: if the flags input specifies illegal or invalid settings. • PROTOCOL_ERROR: if the command is used to read updates from a disabled sensor. See Section 3.1.4 for more status code definitions. If this is an asynchronous call, then the returned status code pertains to this command itself, and any error that occurs during the actual sensor read operation is reported subsequently with the SENSOR_READING_COMPLETE delayed response.
SENSOR_READING readings[N]	An array of sensor readings of format described in Sensor Reading Descriptor, where N is: • 1 for sensors which measure scalar values. • the number of sensor axes for sensors which report values along axes. All axes should be reported in order.

Sensor Reading Descriptor

The SENSOR_READING structure provides the sensor readings and the timestamp when they were collected.

int32 sensor_value_low	Lower 32 bits of the sensor value. This value is invalid if an error status is returned.
int32 sensor_value_high	Higher 32 bits of the sensor value. This value is invalid if an error status is returned.
uint32 timestamp_low	Lower 32 bits of the timestamp when the sample was captured. If no timestamp is collected, this field should be set to 0.

uint32 timestamp_high	Higher 32 bits of the timestamp when the sample was captured. If no timestamp is collected, this field should be set to 0.

3.7.2.13 SENSOR_CONTINUOUS_UPDATE_NOTIFY

This command allows the agent to request notifications from the platform if a sensor has values to report. These notifications are sent using the SENSOR_UPDATE command which is described in Section 3.7.4.2.

The SENSOR_UPDATE notification is sent only when the sensor is enabled. This notification could be generated at every sensor update interval or whenever the sensor values change by a specific margin depending on the sensor characteristics.

The sensor must support continuous update notifications for this command to be used.

If no sensor in the platform supports continuous update notifications, this command can be omitted.

The PROTOCOL_MESSAGE_ATTRIBUTES command, that is described in Section 3.2.2.4, can be used to determine whether this command is implemented.

On initial boot of an agent, by default, these notifications must be disabled from being sent to that agent.

message_id: `0xB`
protocol_id: `0x15`
This command is optional.

Parameters

Name	Description
uint32 sensor_id	Identifier for the sensor to be read.
uint32 notify_enable	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Notify enable: • If this value is 0, the platform does not send any SENSOR_UPDATE notifications to the agent. • If this value is set to 1, the platform sends SENSOR_UPDATE notifications to the agent. See Section 3.7.4.2 for more details about the SENSOR_UPDATE notification.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if sensor_id does not point to a valid domain. • NOT_SUPPORTED: if continuous update notifications are not supported for the indicated sensor. • INVALID_PARAMETERS: if notify_enable specifies illegal or unimplemented options. See Section 3.1.4 for more status code definitions.

3.7.2.14 SENSOR_NAME_GET

This command returns the extended name associated with a specific sensor.

message_id: 0xC
protocol_id: 0x15
This command is mandatory only if extended sensor name is supported.
Extended sensor name is supported when Bit[29] of sensor_attributes_low field of SENSOR_DESCRIPTION_GET command is set to 1. Refer to Section 3.7.2.5 for more details.

Parameters

Name	Description
uint32 sensor_id	Identifier for the sensor.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid sensor extended name is returned. • NOT_FOUND: if sensor_id pertains to a non-existent sensor. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint8 name[64]	Null-terminated UTF-8 format string of up to 64 bytes in length describing the sensor extended name.

3.7.2.15 SENSOR_AXIS_NAME_GET

This command returns the extended name associated with a specific sensor axis. On success, it returns an array of axis name descriptors for the sensor as described in Axis Name Descriptor.

Sometimes it might not be possible to return all the axis name descriptors with just one call. The size of the array returned depends on the number of return values a given transport can support. To solve this problem, the interface allows multiple calls. It also returns the number of remaining axis name descriptors.

The sensor axis name descriptors returned by this call should be reported in the normative order of axes. For example, a triaxial accelerometer should return its axis name descriptors ordered as x, y, and z.

message_id: 0xD
protocol_id: 0x15
This command is mandatory only if extended sensor axis name is supported.
Extended sensor axis name is supported when Bit[9] of axis_attributes_low field of SENSOR_AXIS_DESCRIPTION_GET command is set to 1. Refer to Section 3.7.2.6 for more details.

Parameters

Name	Description
uint32 sensor_id	Identifier for the sensor.
uint32 axis_id	Index of the first sensor axis to be read in the sensor axis name descriptor array.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if valid axis name descriptors were returned. • NOT_FOUND: if sensor_id does not point to a valid sensor or axis_id does not point to a valid axis. • NOT_SUPPORTED: if the sensor does not report values along axis. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:26]** Number of remaining axis name descriptors. **Bits[25:6]** Reserved, must be zero. **Bits[5:0]** Number of axis name descriptors that are returned by this current call.
AXIS_NAME_DESC desc[N]	An array of axis name descriptors, of format described in Axis Name Descriptor.

Axis Name Descriptor

The AXIS_NAME_DESC structure describes the sensor axis extended names.

uint32 axis_id	Identifier for the axis of the sensor.
uint8 name[64]	A NULL terminated UTF-8 format string of up to 64 bytes in length describing the axis extended name. It is recommended that the name ends with '_' followed by the axis of the sensor in uppercase. For example, the name for the x-axis of a triaxial accelerometer could be "acc_X" or "_X".

3.7.3 Delayed Responses

3.7.3.1 SENSOR_READING_COMPLETE

This response is the delayed response to an asynchronous SENSOR_READING_GET command issued by an agent. When the platform determines that there are certain failure conditions in the sensor itself, such as a fault in the sensor hardware or related circuitry or logic, it returns HARDWARE_ERROR to report that condition to the caller. Other errors apply to the interface itself and are enumerated in Section 3.1.4.

message_id: `0x6`
protocol_id: `0x15`
This response is mandatory and is generated if the caller used the asynchronous method to read the sensor.

Return Values

Name	Description
int32 status	An appropriate status code, as described in Section 3.1.4.
uint32 sensor_id	Identifier for the sensor.
SENSOR_READING readings[N]	An array of sensor readings of format described in Sensor Reading Descriptor, where N is the number of sensor axes. All axes should be reported in order.

3.7.4 Notifications

3.7.4.1 SENSOR_TRIP_POINT_EVENT

This notification is issued by the platform when a sensor crosses a specific trip point that the agent had requested event notification for, by using the SENSOR_TRIP_POINT_CONFIG command.

The platform might read sensors periodically using polling, or program sensors to generate interrupts on trip points, depending on implementation. If the sensor value changes such that it crosses several trip-points between successive reads by the platform, then the platform might minimally send only one notification to the agent to represent the multiple cross-over condition.

Message_id: `0x0`
protocol_id: `0x15`
This notification is optional.

Return Values

Name	Description
uint32 agent_id	Refers to the agent that caused this event. For the current version of the specification, this field is set to 0 to indicate that the platform is the generator of all sensor events.
uint32 sensor_id	Identifier for the sensor that has tripped.
uint32 trip_point_desc	**Bits[31:17]** Reserved. **Bit[16]** Direction. • If set to 1, indicates that the trip point was reached or crossed in the positive direction. • If set to 0, indicates that the trip point was reached or crossed in the negative direction. **Bits[15:8]** Reserved for future use. **Bits[7:0]** trip_point_id. The identifier for the trip point that was crossed or reached.

3.7.4.2 SENSOR_UPDATE

The SENSOR_UPDATE notification is issued by the platform to provide a sensor reading.

This notification is sent to an agent which had requested continuous update notifications from a sensor using SENSOR_CONTINUOUS_UPDATE_NOTIFY command.

The SENSOR_UPDATE notification is sent only when the sensor is enabled. This notification could be generated at every sensor update interval or whenever the sensor values change by a specific margin depending on the sensor characteristics.

The notification is generated at a rate no faster than the configured sensor update period. The platform is allowed to generate the notification at a rate slower than the configured sensor update period if the agent or the platform cannot cope with the rate of generation of notifications due to hardware constraints. However, the platform should always provide the latest sensor values in the SENSOR_UPDATE notification.

The platform does not need to implement this notification if no sensors support continuous update notifications.

message_id: 0x1
protocol_id: 0x15
This notification is mandatory if at least one sensor supports continuous update notifications.

Return Values

Name	Description
uint32 agent_id	Refers to the agent that caused this event. For the current version of the specification, this field is set to 0 to indicate that the platform is the generator of all sensor events.
uint32 sensor_id	Identifier for the sensor.
SENSOR_READING readings[N]	An array of sensor readings of format described in Sensor Reading Descriptor, where N is: • 1 for sensors which measure scalar values. • the number of sensor axes for sensors which report values along axes. All axes should be reported in order.

3.7.5 Sensor Values Shared Memory

Note: Statistics capability has been marked for deprecation and is being replaced by system telemetry.

Optionally, the platform might provide sensor values through the shared memory region that is associated with the sensor management protocol. Whether support is present is indicated by the PROTOCOL_ATTRIBUTES command, which is described in Section 3.7.2.3. This command also provides the address and the size of the shared memory region. The memory must be accessible from the Non-secure world, and OSPM must map it as non-cached normal memory or device memory.

Accessing multi-word values might cause races between platform write accesses and the read accesses by agents in the system. This problem and its solution are described in Section 3.3.4.

The format of the sensor shared memory region is described in Table 20.

Table 20: Sensor shared memory region

Field	Byte Length	Byte Offset	Description
Signature	0x4	0x0	0x53454E53 (‘SENS’).
Revision	0x2	0x4	For this revision, this value must be 0x1.
Attributes	0x2	0x6	For this revision, this value must be zero.
Number of sensors	0x2	0x8	Number of sensors.
Reserved	0x2	0xA	Must be zero.
Match Sequence	0x4	0xC	The match sequence populated by the platform as described in Section 3.3.4.1

Field	Byte Length	Byte Offset	Description
Sensor domain offset array	0x4 × Number of sensors	0x10	For each sensor, this array provides a 4-byte offset, from the start of the shared memory area, to the memory location of the sensor value data entry in the data section. The array is indexed by sensor_id. The sensor value data entry format is described in Table 21. A value of 0 indicates that the sensor value is not reported through shared memory.
Sensor values data section	--	--	This area must start at an offset of 0x10 + 0x4 × (Number of sensors), or higher.

The sensor values data section contains entries for each sensor value. The format for each sensor value data entry is described in Table 21.

Table 21: Sensor value data entry

Field	Byte Length	Byte Offset	Description
Sensor Value	0x10 × N	0x0	The sensor value is stored on a 64-bit aligned boundary in the format specified by Sensor Reading Descriptor where N is: • 1 for sensors which measure scalar values. • the number of sensor axes for sensors which report values along axes. All axes should be reported in order.

Accessing statistics can cause races between platform write accesses and agent read accesses. This problem and its solution are described in Section 3.3.4.1.

3.8 Reset domain management protocol

This protocol is intended for control of reset capable domains in the platform. The reset management protocol provides commands to:

- Describe the protocol version.
- Discover the attributes and capabilities of the reset domains in the system.
- Reset a given domain.
- Receive notifications when a given domain is reset.

3.8.1 Reset domain management protocol background

Devices that can be collectively reset through a common reset signal constitute a reset domain. A reset domain can be reset autonomously or explicitly. When autonomous reset is chosen, the firmware is responsible for taking the necessary steps to reset the domain and to subsequently bring it out of reset. When explicit reset is chosen, the caller must specifically assert and then de-assert the reset signal by issuing two separate RESET commands.

Reset State encoding for reset domains is described below in Table 22.

Table 22: Reset State Parameter Layout

Bit field	Description
31	Reset Type If set to 0, indicates Architectural Reset. If set to 1, indicates IMPLEMENTATION defined Reset.
30:0	Reset ID

The two distinct reset types possible are architectural reset and IMPLEMENTATION defined reset. Reset Types and Reset IDs are described in Table 23.

Table 23: Reset Type and Reset ID Description

Reset Type	Reset ID	Description
Architectural Reset	0x0	COLD_RESET. Full loss of context of all devices in the domain.
	0x1-0x7FFFFFFF	Reserved for future use. Lower values indicate greater context loss.
IMPLEMENTATION defined Reset	0x0-0x7FFFFFFF	IMPLEMENTATION DEFINED Resets. All values represent resets that result in varying levels of context loss. Lower values indicate greater context loss.

Reset domains are not the same as power domains, although they can be the same. There could be multiple reset domains within a given power domain. There could also be reset domains that straddle multiple power domains.

Resets might impose the requirement that devices in the affected reset domain are in a state of quiescence before the reset is issued. Support for such quiescence might be provided by the reset domain. In the absence of such a support, it is the calling agent's responsibility to ensure quiescence prior to invocation of the reset.

Protocol commands take integer identifiers to identify the reset domain they apply to. The identifiers are sequential and start from 0.

For all combinations of reset domains and agents, platform policy dictates which agents can access which reset domains. This policy is guided by the required agent functionality while maintaining the security, confidentiality, and integrity of other agents in the system. It is recommended not to share the same reset domain among peripherals which are used by different agents in the system concurrently.

3.8.2 Commands

3.8.2.1 PROTOCOL_VERSION

On success, this command returns the version of this protocol. For this version of the specification, the value returned must be `0x30001`, which corresponds to version 3.1.

message_id: `0x0`
protocol_id: `0x16`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this must be `0x30001`.

3.8.2.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: `0x10`
protocol_id: `0x16`
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.8.2.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: 0x1
protocol_id: 0x16
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 attributes	**Bits[31:16]** Reserved, must be zero. **Bits[15:0]** Number of reset domains.

3.8.2.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

message_id: 0x2
protocol_id: 0x16
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is not provided by this platform implementation. See Section 3.1.4 for more status code definitions.
uint32 attributes	Reserved, must be zero.

3.8.2.5 RESET_DOMAIN_ATTRIBUTES

This command returns attributes of the reset domain specified in the command.

message_id: 0x3
protocol_id: 0x16
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the reset domain.

Return values	
Name	**Description**
int32 status	One of, but not limited to, the following: • SUCCESS: if valid reset domain attributes were returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.
uint32 attributes	**Bit[31]** Asynchronous reset support. • Set to 1 if this domain can be reset asynchronously. • Set to 0 if this domain can only be reset synchronously. **Bit[30]** Reset notifications support. • Set to 1 if reset notifications are supported for this domain. • Set to 0 if reset notifications are not supported for this domain. **Bit[29]** Extended reset domain name. • If set to 1, the reset domain name is greater than 16 bytes. The extended reset domain name is provided by RESET_DOMAIN_NAME_GET command which is described in Section 3.8.2.8. • If set to 0, extended reset domain name is not supported. **Bits[28:0]** Reserved, must be zero.
uint32 latency	Maximum time (in microseconds) required for the reset to take effect on the given domain. A value of 0xFFFFFFFF indicates this field is not supported by the platform.
uint8 name[16]	Null-terminated ASCII string of up to 16 bytes in length describing the reset domain name. When Bit[29] of attributes field is set to 1, this field contains the lower 15 bytes of the NULL terminated reset domain name.

3.8.2.6 RESET

This command allows an agent to reset the specified reset domain. If the reset request is issued as an asynchronous call, the platform must return immediately upon receipt of the request. The platform might need to ensure that the domain and all dependent logic have reached a state of quiescence before performing the actual reset, although this is not mandatory.

When the reset is done, the platform should then send a RESET_COMPLETE delayed response, described in Section 3.8.3.1. The platform has the option to inform agents other than the caller of the reset incident, using the RESET_ISSUED notification that is described in Section 3.8.4.1.

message_id: `0x4`
protocol_id: `0x16`
This command is mandatory.

Parameters	
Name	**Description**
uint32 domain_id	Identifier for the reset domain.

uint32 flags	This parameter allows the agent to specify additional conditions and requirements specific to the request, and has the following format: **Bits[31:3]** Reserved, must be zero. **Bit[2]** Async flag. Only valid if Bit[0] is set to 1. • Set to 1 if the reset must complete asynchronously. • Set to 0 if the reset must complete synchronously. **Bit[1]** Explicit signal. This flag is ignored when Bit[0] is set to 1. • Set to 1 to explicitly assert reset signal. • Set to 0 to explicitly de-assert reset signal. **Bit[0]** Autonomous Reset action. • Set to 1 if the reset must be performed autonomously by the platform. • Set to 0 if the reset signal shall be explicitly asserted and de-asserted by the caller.
uint32 reset_state	The reset state being requested. The format of this parameter is specified in Table 22.

Return values	
Name	**Description**
int32 status	One of, but not limited to, the following: • SUCCESS: if the operation was successful. • NOT_FOUND: if the reset domain identified by domain_id does not exist. • INVALID_PARAMETERS: if an illegal or unsupported reset state is specified or if the flags field is invalid. • GENERIC_ERROR: if the operation failed, for example if there are other active users of the reset domain. • DENIED: if the calling agent is not allowed to reset the specified reset domain. See Section 3.1.4 for more status code definitions.

3.8.2.7 RESET_NOTIFY

This command allows the caller to request notifications from the platform when a reset domain has been reset. If reset has been explicitly signaled, the platform generates this notification when the reset signal has been asserted. These notifications are sent using the RESET_ISSUED notification, which is described in Section 3.8.4.1.

Notification support is optional, and PROTOCOL_MESSAGE_ATTRIBUTES must be used to discover whether this command is implemented.

These notifications must be disabled by default during initial boot of the platform.

message_id: `0x5` protocol_id: `0x16` This command is optional.

Parameters	
Name	**Description**

uint32 domain_id	Identifier for the reset domain.
uint32 notify_enable	**Bits[31:1]** Reserved must be zero. **Bit[0]** Notify enable. This bit can have one of the following values: • 1, which indicates that the platform should send RESET_ISSUED notifications to the calling agent when the domain is reset. • 0, which indicates that the platform should not send any RESET_ISSUED notifications to the calling agent.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if domain_id does not point to a valid domain. • INVALID_PARAMETERS: if notify_enable specifies values that are either illegal or incorrect. See Section 3.1.4 for more status code definitions.

3.8.2.8 RESET_DOMAIN_NAME_GET

This command returns the extended name associated with a specific reset domain.

message_id: `0x6`
protocol_id: `0x16`
This command is mandatory only if extended reset domain name is supported.
Extended reset domain name is supported when Bit[29] of attributes field of RESET_DOMAIN_ATTRIBUTES command is set to 1. Refer to Section 3.8.2.5 for more details.

Parameters

Name	Description
uint32 domain_id	Identifier for the reset domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid reset domain extended name is returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint8 name[64]	Null-terminated ASCII string of up to 64 bytes in length describing the reset domain extended name.

3.8.3 Delayed Responses

3.8.3.1 RESET_COMPLETE

The platform sends this delayed response to the caller that requested an asynchronous reset of the specified domain.

message_id: 0x4
protocol_id: 0x16
This command is optional.

Parameters

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if reset was successful. • GENERIC_ERROR: if the operation failed, for example if there were other users of the reset domain, or if the domain could not be brought to a state of quiescence preparatory to the reset. Other vendor-specific errors can also be generated depending on the implementation. See Section 3.1.4 for more status code definitions.
uint32 domain_id	Identifier for the reset domain.

3.8.4 Notifications

3.8.4.1 RESET_ISSUED

The platform sends this notification to an agent that has registered to receive notifications when the reset domain identified by domain_id has been reset. The notification might not be received if the agent is affected as a result of the reset.

message_id: 0x0
protocol_id: 0x16
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent that caused the reset of the domain.
uint32 domain_id	Identifier of the reset domain.
uint32 reset_state	The reset state issued on the domain. The format of this parameter is specified in Table 22.

3.9 Voltage domain management protocol

This protocol is intended for the management of configuration and voltage levels of voltage domains. The voltage domain management protocol provides commands to:

- Describe the protocol version.
- Discover implementation attributes.
- Discover the voltage levels supported by a domain.
- Get the configuration and voltage level of a domain.
- Set the configuration and voltage level of a domain.

3.9.1 Voltage domain management protocol background

In this document, a voltage domain is defined as a group of components that are supplied by a single voltage source. A voltage domain is different from a power domain which is a collection of elements within a voltage domain that share common power control. A voltage domain can be partitioned into one or more power domains.

Voltage domain protocol can be used for managing voltage supply to standard components like embedded multi-media cards, USB and others which specify a standard operating voltage.

Voltage domains have the following properties:

- They can include one or more devices.
- Their voltage supply can be scaled or removed for power management.

Protocol commands take integer identifiers to identify the voltage domain they apply to. The identifiers are sequential and start from 0.

Note

Voltage domain management protocol should not be used:

- to control the performance of application processors or devices.
- if one or more clock rates need to be adjusted as a direct consequence of changing the voltage level of a voltage domain.

The Performance domain management protocol should be used in all the above cases.

An implementation can include devices that are intended for use only by Secure entities in the system such as a trusted OS. Voltage domains for such devices must be managed through Secure channels.

Non-secure channels can be used to manage voltage domains for devices which are not used by Secure or Root entities in the system.

In a multi-agent system with multiple domains, several scenarios are possible:

- A voltage domain is exclusive to an agent.
- A voltage domain can be shared by multiple agents.

In both these cases, the agents can coordinate with the platform to access voltage domains, and to perform voltage management of the domains. For all combinations of voltage domains and agents, platform policy dictates which agents can access which voltage domains, and whether a voltage domain is shared or exclusive. This policy is guided by the required agent functionality while maintaining the security, confidentiality, and integrity of other agents in the system It is recommended not to share the same voltage domain among agents which require different voltage levels from that voltage domain concurrently.

Voltage domains can operate in different modes are described below in Table 24. It is not necessary for a voltage domain to support all the modes of operation.

Table 24: Voltage Domain Mode Parameter Layout

Bit field	Description
3	Mode Type If set to 0, indicates Architectural Mode. If set to 1, indicates IMPLEMENTATION defined Mode.
2:0	Mode ID

The two distinct voltage domain mode types possible are architectural mode and IMPLEMENTATION defined mode. Mode Types and Mode IDs are described in Table 25.

Table 25: Mode Type and Mode ID Description

Mode Type	Mode ID	Description
Architectural Mode	0x0	OFF. Voltage supply to the domain is disabled.
	0x1-0x6	Reserved for future use.
	0x7	ON. Voltage supply to the domain is active. The components supplied by the voltage domain can operate normally.
IMPLEMENTATION defined Mode	0x0-0x7	IMPLEMENTATION defined Modes.

3.9.2 Commands

3.9.2.1 PROTOCOL_VERSION

On success, this command returns the protocol version. For this version of the specification, the return value must be 0x20001, which corresponds to version 2.1.

message_id: 0x0
protocol_id: 0x17
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this value must be 0x20001.

3.9.2.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: `0x10`
protocol_id: `0x17`
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.9.2.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: `0x1`
protocol_id: `0x17`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 attributes	**Bits[31:16]** Reserved, must be zero. **Bits[15:0]** Number of voltage domains.

3.9.2.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

message_id: `0x2`
protocol_id: `0x17`
This command is mandatory.

Parameters

Name	Description

uint32 message_id	message_id of the message.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is invalid or not implemented. See Section 3.1.4 for more status code definitions.
uint32 attributes	Flags that are associated with a specific command in the protocol. In the current version of the specification, this value is always 0.

3.9.2.5 VOLTAGE_DOMAIN_ATTRIBUTES

This command returns the attribute flags associated with a specific voltage domain.

message_id: `0x3`
protocol_id: `0x17`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the domain. Domain identifiers are limited to 16 bits, and the upper 16 bits of this field are ignored by the platform.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid voltage domain attributes were returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.
uint32 attributes	**Bit[31]** Asynchronous voltage level set support. • Set to 1 if the voltage level of this domain can be set asynchronously. • Set to 0 if the voltage level of this domain can only be set synchronously. **Bit[30]** Extended voltage domain name. • If set to 1, the voltage domain name is greater than 16 bytes. The extended voltage domain name is provided by VOLTAGE_DOMAIN_NAME_GET command which is described in Section 3.9.2.11. • If set to 0, extended voltage domain name is not supported. **Bits[29:0]** Reserved, must be zero.
uint8 name[16]	Null-terminated ASCII string of up to 16 bytes in length describing the voltage domain name. When Bit[30] of attributes field is set to 1, this field contains the lower 15 bytes of the NULL terminated voltage domain name.

3.9.2.6 VOLTAGE_DESCRIBE_LEVELS

This command allows the agent to ascertain the voltage levels supported by a voltage domain. On success, the command returns an array, which contains a number of voltage level entries. Sometimes it might not be possible to return the whole array with just one call. To solve this problem, the interface allows multiple calls. It also returns the number of remaining voltage levels. The size of the array returned depends on the number of return values a given transport can support.

Voltage domains can support many voltage levels and sometimes individually describing each voltage level might be too onerous. In such cases, the command can return only the lowest voltage level, the highest voltage level and the step size between two successive voltage levels that the voltage domain supports.

The voltage levels returned by this call should be in numeric ascending order.

For an example of using this kind of API, see Section 3.5.6.8.

message_id: `0x4`
protocol_id: `0x17`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the voltage domain.
uint32 level_index	Index of the first voltage level to be described in the return voltage array.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the voltage levels are returned successfully. • NOT_FOUND: if the domain identified by domain_id does not exist. • OUT_OF_RANGE: if the level_index is outside of valid range. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to get the voltage levels supported for this voltage domain. An example would be if this voltage domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.
uint32 flags	Descriptor for the voltage levels supported by this domain. **Bits[31:16]** Number of remaining voltage levels. This field should be 0 if Bit[12] is 1. **Bits[15:13]** Reserved, must be zero. **Bit[12]** Return format: • If this bit is set to 1, the Voltage Array is a triplet that constitutes a segment in the following form: – voltage[0] is the lowest voltage level that the domain supports. – voltage[1] is the highest voltage level that the domain supports. – voltage[2] is the step size between two successive voltage levels that the domain supports. • If this bit is set to 0, each element of the Voltage Array represents a discrete voltage level that the voltage domain supports. **Bits[11:0]** Number of voltage levels that are returned by this call. This field should be 3 if Bit[12] is 1.

int32 voltage [N]	Voltage Array expressed in microvolts (uV): If Bit[12] of the flags field is set to 0, each array entry represents a discrete voltage level. If Bit[12] of the flags field is set to 1, then each entry is a member of a segment {lowest voltage level, highest voltage level, step size} as described above. N is specified by Bits[11:0] of flags field.

3.9.2.7 VOLTAGE_CONFIG_SET

This command allows an agent to set the configuration of a voltage domain. It allows to enable or disable the domain.

message_id: `0x5`
protocol_id: `0x17`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the voltage domain.
uint32 config	**Bits[31:4]** Reserved, must be zero. **Bits[3:0]** Mode: The operating mode the voltage domain should be set to, as described in Table 24.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the voltage domain configuration has been set successfully. • NOT_FOUND: if the voltage domain identified by domain_id does not exist. • INVALID_PARAMETERS: if the requested configuration is not supported by the voltage domain. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to set the configuration of this voltage domain. An example would be if this voltage domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.

3.9.2.8 VOLTAGE_CONFIG_GET

This command allows the calling agent to request the configuration of a voltage domain.

message_id: `0x6`
protocol_id: `0x17`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the voltage domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the voltage domain configuration was successfully returned. • NOT_FOUND: if domain_id does not point to a valid voltage domain. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to get the configuration of this voltage domain. An example would be if this voltage domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.
uint32 config	**Bits[31:4]** Reserved, must be zero. **Bits[3:0]** Mode: The operating mode of the voltage domain, as described in Table 24.

3.9.2.9 VOLTAGE_LEVEL_SET

This command allows an agent to set the voltage level of a voltage domain.

message_id: `0x7`
protocol_id: `0x17`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the voltage domain.
uint32 flags	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Async flag: • Set to 1 if the voltage level is to be set asynchronously. In this case the call is completed with VOLTAGE_LEVEL_SET_COMPLETE message. For more details, see Section 3.9.3.1. A SUCCESS return code in this case indicates that the platform has successfully queued this command • Set 0 to if the voltage level is to be set synchronously. In this case, the call will return when the voltage level has been set.
int32 voltage_level	The voltage level, in microvolts (uV), to set the domain to.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if the voltage domain has been set to the desired level. • NOT_FOUND: if the voltage domain identified by domain_id does not exist. • INVALID_PARAMETERS: if the requested voltage level is not supported by the voltage domain. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to set the voltage level of this voltage domain. An example would be if this voltage domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.

3.9.2.10 VOLTAGE_LEVEL_GET

This command allows the calling agent to request the current voltage level of a voltage domain.

Note

It is possible for the voltage_level value returned by this command to be stale by the time the command completes, as another voltage level change request could have been initiated and completed in the interim.

message_id: `0x8`
protocol_id: `0x17`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the voltage domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the voltage level of the domain was returned successfully. • NOT_FOUND: if domain_id does not point to a valid voltage domain. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to get the voltage level of this voltage domain. An example would be if this voltage domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.
int32 voltage_level	The voltage level, in microvolts (uV), that the domain is set to.

3.9.2.11 VOLTAGE_DOMAIN_NAME_GET

This command returns the extended name associated with a specific voltage domain.

message_id: 0x9
protocol_id: 0x17
This command is mandatory only if extended voltage domain name is supported.
Extended voltage domain name is supported when Bit[30] of attributes field of VOLTAGE_DOMAIN_ATTRIBUTES command is set to 1. Refer to Section 3.9.2.5 for more details.

Parameters

Name	Description
uint32 domain_id	Identifier for the voltage domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid voltage domain extended name is returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint8 name[64]	Null-terminated ASCII string of up to 64 bytes in length describing the voltage domain extended name.

3.9.3 Delayed Responses

3.9.3.1 VOLTAGE_LEVEL_SET_COMPLETE

The platform sends this delayed response to the caller that requested to set the voltage of a domain asynchronously.

message_id: 0x7
protocol_id: 0x17
This command is optional.

Parameters

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the voltage of the domain has been set successfully. Other vendor-specific errors can also be generated depending on the implementation. See Section 3.1.4 for more status code definitions.
uint32 domain_id	Identifier for the voltage domain.
int32 voltage_level	The voltage level, in microvolts (uV), that the domain has been set to.

3.10 Power capping and monitoring protocol

This protocol is intended for controlling and monitoring the power consumption of power capping domains. The power capping and monitoring protocol provides commands to:

- Describe the protocol version.
- Discover implementation attributes.
- Set and get the power cap for a domain.
- Optionally monitor the consumed power for the domain.
- Optionally request for notifications when the enforced power cap or average power measurements for a domain is changed.

3.10.1 Power capping and monitoring protocol background

A power capping domain is defined as a group of platform components whose power consumption is managed and monitored as a single entity. A power capping domain is different from a power domain (which is a collection of elements that share a common power control), or a voltage domain (which is a collection of elements that are supplied by a single voltage source). A power capping domain can be partitioned into one or more power domains or voltage domains. It is possible to have a power capping domain which only offers monitoring of domain power consumption, instead of offering both control and monitoring of domain power consumption.

Figure 6 shows an example of power capping domains.

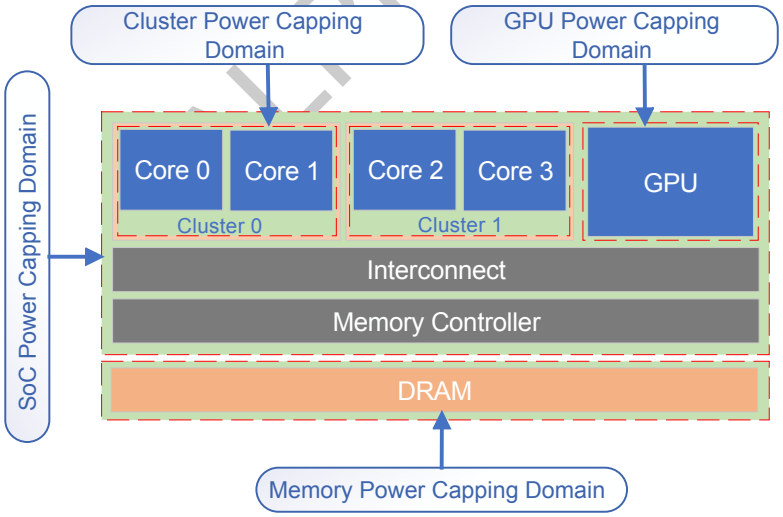

Figure 6: Example power capping domains

A power capping domain can be completely contained within another power capping domain. Such domains have a parent-child relationship, where the parent domain contains the child domain. In all other cases, power capping domains should be mutually exclusive. When domains have a parent-child relationship, any disabling of power caps on the parent domain do not automatically disable power caps on its child domains. In Figure 6, processor clusters 0 and 1 are two distinct power capping domains. They are also children of the SoC power capping domain. Similarly, the GPU is part of the SoC domain as well as the GPU domain. However, it is not possible for Core 2 or any other component to be a part

of both Cluster 0 power capping domain as well as Cluster 1 power capping domain since Cluster 0 and Cluster 1 do not have a parent child relationship.

Some commonly used terms in this protocol are:

- **Power Cap**: The value (in power units) which the power consumption of a power capping domain should be limited to.

  Power units can be specified in microwatts (uW), milliwatts(mW) or on an abstract linear integer scale. The implementation can choose what this scale represents. For example, in some systems, the values in the scale might represent actual power values, while in others they might represent a percentage of the maximum possible power consumption of the domain. In all cases, the scale must be linear, meaning that a value of 2X consumes twice the power as compared to a value of X.

- **Capping Averaging Interval (CAI)**: The rolling time interval over which the average power consumption of a power capping domain is limited. CAI is expressed in microseconds (us). The platform may choose not to report CAI.

- **Power Measurement Averaging Interval (MAI)**: The rolling time interval over which the average power consumption of a power capping domain is measured for the purpose of reporting. MAI is expressed in microseconds (us). The platform may choose not to report MAI.

  Note: Version 2.0 of the Power Capping and Monitoring Protocol used the term PAI to refer to both CAI and MAI. Usage of PAI has been deprecated.

- **Concurrent Power Capping (CPC)**: A feature where a power capping domain supports multiple power limits applied concurrently over different CAIs or over different ranges of CAIs. A domain supporting CPC can only enforce a limited number of power limits concurrently. CAI ranges associated with concurrently enforced power limits cannot overlap with each other. For example, a power capping domain with CPC could support three power limits, one limit each for a CAI range of 1-10 us, 1000-5000 us, and 1-5 second respectively. Domains which do not support CPC only support a single power limit over a CAI or range of CAIs.

- **Concurrent Power Limit identifier (CPLi)**: An identifier used to represent each power limit associated with a CAI or range of CAIs, which can be enforced concurrently when CPC is supported. Concurrent Power Limit identifiers are represented by sequential integer identifiers starting from 0. The Concurrent Power Limit identifiers must be ordered inversely with decreasing CAI values. For example, if a power capping domain supports four Concurrent Power Limit identifiers, they must be ordered as represented in Table 26. CAI ranges represented in the table are for illustration only.

Table 26: Concurrent Power Limits example

CPLi, where i = {0, 1, 2, 3}	CAI	Name
CPL0	90 - 300 seconds	“CPL_sus”
CPL1	1 - 40 seconds	“CPL_med”
CPL2	4 - 100 milliseconds	“CPL_short”
CPL3	10 microseconds	“CPL_crit”

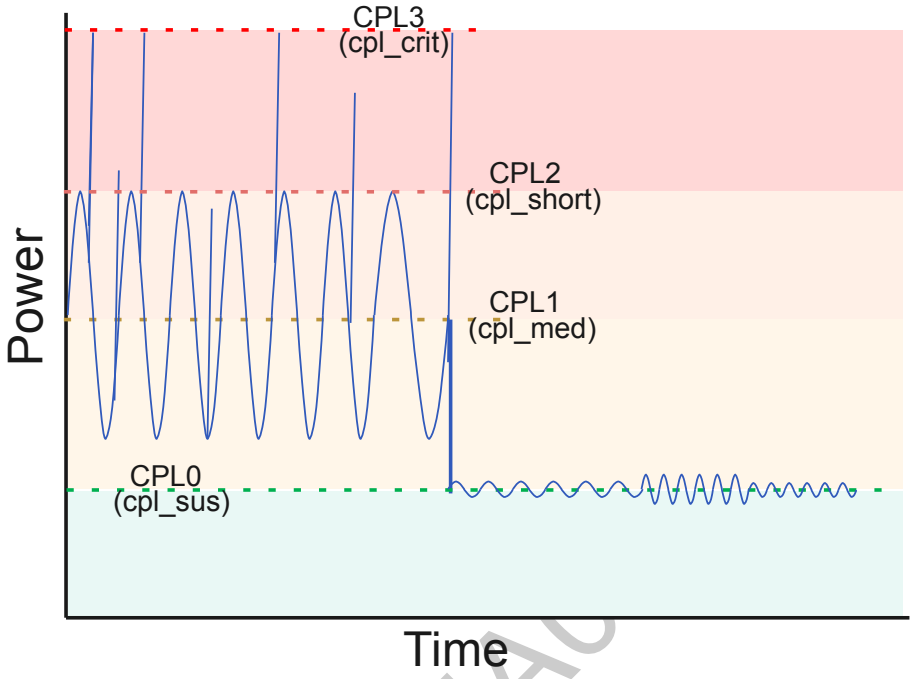

Figure 7: Example power curve of a power capping domain with four concurrent power limits (CPL0-CPL3)

Protocol commands take integer identifiers to describe the power capping domain they apply to. The identifiers are sequential and start from 0.

In a multi-agent system with multiple domains, several scenarios are possible:

- A power capping domain is exclusive to an agent.
- A power capping domain is shared among multiple agents.

In both the above cases, the agents can coordinate with the platform to monitor and manage power capping domains. For all combinations of power capping domains and agents, platform policy determines which agents can access which domains, and whether a power capping domain is shared or exclusive to an agent. This policy depends on the required agent functionality while maintaining the security, confidentiality, and integrity of other agents in the system. It is the responsibility of the platform to resolve requests from different agents for the same domain.

A system can include devices that are dedicated exclusively to Secure entities in the system such as a trusted OS. These devices are not available for agents in the Non-secure Security state to use. Power capping domains which exclusively comprise such devices must be managed through Secure channels.

There are typically two agents that are expected to use the power cap protocol:

- OSPM agent running on the Application PE.
- Management Agent on the Satellite Management Controller.

Each of these agents have their own “view” of the power cap and monitoring protocol in terms of attributes and command availability for a power capping domain. It is the responsibility of the platform to resolve power capping requests from these two agents and apply a suitable policy. An example policy could be to apply the highest power cap requested. Platform power capping policy is implementation defined and out of the scope of this specification.

Note

Power Capping can be exploited for side-channel attacks. It is recommended that this protocol is enabled only after due consideration of the potential possibility and mechanism of such an attack. It is also recommended that the power cap and the power measurement granularity is restricted on production systems after considering the worst-case data dependent variation of the power consumption of the power capping domain. Additional defense mechanisms might include, but are not restricted to, the introduction of random noise margins to power measurements and certificate-based enablement of relevant platform functionality. It is the responsibility of the platform to ensure that this protocol is not exploitable for attacks. The exact mechanism used to identify the deployment parameters of this protocol is beyond the scope of this specification.

3.10.2 FastChannels

This section describes the properties of FastChannels for Power Capping and Monitoring Protocol.

- Only the following commands are supported over FastChannels:
  - POWERCAP_CAP_SET
  - POWERCAP_CAP_GET
  - POWERCAP_MAI_SET
  - POWERCAP_MAI_GET
  - POWERCAP_CAI_SET
  - POWERCAP_CAI_GET
  - POWERCAP_MEASUREMENTS_GET
- A FastChannel needs to be unique for any combination of power capping domain, Concurrent Power Limit identifier and power capping and monitoring command. It is not necessary for every power capping domain or every Power Capping and Monitoring Commandto support a FastChannel.
- FastChannels are discoverable via the POWERCAP_DESCRIBE_FASTCHANNEL command.
- Doorbell is not supported for POWERCAP_CAP_GET, POWERCAP_CAI_GET and POWERCAP_MAI_GET commands. If FastChannels are implemented for these commands, the last known valid power cap CAI or MAI must always be available over the FastChannel without a doorbell trigger. This property reduces complexity due to latency considerations between doorbell trigger and the availability of return values over the FastChannel. For all other commands, Doorbell support is optional.

For more details on FastChannels, see Section 4.3.

3.10.2.1 Payload Requirements

The payload of a FastChannel must only contain the message-specific parameters and exclude the domain_id. Since a FastChannel is domain_id, cpli, and message_id specific, the domain_id or any other channel-specific and message-specific headers do not need to be included while using a FastChannel.

All FastChannels associated with this protocol return 32 bits of payload. For example, the payload of the POWERCAP_CAP_SET message should be `uint32 power_cap`. In addition, the FastChannel for POWERCAP_MEASUREMENTS_GET returns only the power measured as payload, and the measurement averaging interval is not returned.

3.10.3 Commands

3.10.3.1 PROTOCOL_VERSION

On success, this command returns the protocol version. For this version of the specification, the return value must be `0x30000`, which corresponds to version 3.0.

message_id: 0x0
protocol_id: 0x18
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this value must be 0x30000.

3.10.3.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: 0x10
protocol_id: 0x18
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.10.3.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: 0x1
protocol_id: 0x18
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.

uint32 attributes	**Bits[31:16]** Reserved, must be zero. **Bits[15:0]** Number of power capping domains.

3.10.3.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

This command can be used to inquire if power cap, CAI, and MAI change notifications are supported, by passing POWERCAP_CAP_NOTIFY message identifier to the call. It can also be used to inquire if power measurements change notification is supported, by passing POWERCAP_MEASUREMENTS_NOTIFY message identifier to the call. If the platform returns SUCCESS, then it supports the corresponding notifications. Otherwise, if the platform returns NOT_FOUND, then it is an indication that the notifications are not implemented, or that the notifications are not available to the calling agent. The notification commands are described in Section 3.10.3.15 and Section 3.10.3.16.

message_id: `0x2`
protocol_id: `0x18`
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is invalid or not implemented. See Section 3.1.4 for more status code definitions.
uint32 attributes	Flags that are associated with a specific command in the protocol. **Bits[31:1]** Reserved, must be zero. **Bit[0]** FastChannel Support. • Set to 1 if there is at least one dedicated FastChannel available for this message. • Set to 0 if there are no FastChannels available for this message.

3.10.3.5 POWERCAP_DOMAIN_ATTRIBUTES

This command returns the attributes associated with a specific power capping domain.

message_id: `0x3`
protocol_id: `0x18`
This command is mandatory.

Parameters

Name	Description
uint32 domain_id	Identifier for the domain. Domain identifiers are limited to 16 bits, and the upper 16 bits of this field are ignored by the platform.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid domain attributes were returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.
uint32 attributes	**Bit[31]** MAI change notifications support. • Set to 1 if MAI change notifications are supported for this power capping domain. • Set to 0 if MAI change notifications are not supported for this power capping domain. **Bit[30]** Power measurements change notification support. • Set to 1 if power measurements change notification is supported for this power capping domain. • Set to 0 if power measurements change notification is not supported for this power capping domain. **Bit[29]** Asynchronous power cap set support. • Set to 1 if the power cap of this domain can be set asynchronously. • Set to 0 if the power cap of this domain can only be set synchronously. **Bit[28]** Extended domain name. • If set to 1, the power capping domain name is greater than 16 bytes. The extended domain name is provided by POWERCAP_DOMAIN_NAME_GET command which is described in Section 3.10.3.13. • If set to 0, extended power capping domain name is not supported. **Bit[27]** Power Cap configuration support. • If set to 1, this domain supports changing power caps. • If set to 0, this domain does not support changing power caps. Bit[26] cannot be zero when this bit is zero.

	Bit[26] Power Monitoring support. • If set to 1, this domain supports power monitoring. • If set to 0, this domain does not support power monitoring. Bit[27] cannot be zero when this bit is zero. **Bit[25]** MAI configuration support. • If set to 1, this domain supports changing the MAI. • If set to 0, this domain does not support changing the MAI. **Bits[24:23]** Power unit used for capping and monitoring. • If set to `0x2`, the power unit is in milliwatts (mW). • If set to `0x1`, the power unit is in microwatts(uW). • If set to `0x0`, the power unit is in an abstract linear integer scale. All other values are reserved and must not be used. **Bit[22]** FastChannel Support. • Set to 1 if there is at least one FastChannel available for this domain. • Set to 0 if there are no FastChannels available for this domain. **Bit[21]** Power cap change notifications support. • Set to 1 if power cap change notifications are supported for this power capping domain. • Set to 0 if power cap change notifications are not supported for this power capping domain. **Bit[20]** CAI configuration support. • If set to 1, this domain supports changing the CAI • If set to 0, this domain does not support changing the CAI **Bit[19]** CAI change notifications support. • Set to 1 if CAI change notifications are supported for this power capping domain. • Set to 0 if CAI change notifications are not supported for this power capping domain. **Bit[18:15]** Number of concurrent power limits supported by this domain. • This field must be greater than 0 if Bit[27] is set to 1 • CPC is supported if this field is greater than 1. **Bits[14:0]** Reserved, must be zero.
uint8 name[16]	Null-terminated ASCII string of up to 16 bytes in length describing the power capping domain name. When Bit[28] of attributes field is set to 1, this field contains the lower 15 bytes of the NULL terminated domain name.
uint32 min_mai	The minimum supported power measurement averaging interval (MAI), expressed in microseconds. When min_mai and max_mai fields return the same value, it indicates that MAI is not configurable. This field is set to zero if the platform does not report this value.

uint32 max_mai	The maximum supported power measurement averaging interval (MAI), expressed in microseconds. When min_mai and max_mai fields return the same value, it indicates that MAI is not configurable. This field is set to zero if the platform does not report this value.
uint32 mai_step	The step size between two consecutive MAI supported by this power capping domain. This field must not be zero if min_mai and max_mai fields report different values.
uint32 min_power_cap	The minimum power value that can be set as the power cap for this domain. It is illegal for this field to be zero. If CPC is supported, this field references CPL0. When min_power_cap and max_power_cap fields return the same value, it indicates that power cap is not configurable.
uint32 max_power_cap	The maximum power value that can be set as a power cap for this domain. It is illegal for this field to be zero. If CPC is supported, this field references CPL0. When min_power_cap and max_power_cap fields return the same value, it indicates that power cap is not configurable.
uint32 power_cap_step	The step size between two consecutive power cap values supported by this power capping domain. If CPC is supported, this field references CPL0. This field must not be zero if min_power_cap and max_power_cap fields report different values.
uint32 sustainable_power	The maximum sustainable power consumption for this domain under normal conditions considering all known external factors like thermal constraints. All power capping domains should be able to maintain their sustained power level simultaneously. In exceptional circumstances, such as thermal runaway, the platform might not be able to guarantee this level.
uint32 accuracy	The accuracy with which the power is measured and reported in integral multiples of 0.001 percent. For example, a value of 10 means 0.01 percent accuracy. A value of zero indicates that the platform does not report this value.
uint32 parent_id	This field identifies the parent power capping domain. This field is useful in constructing a power capping domain topology when the power capping domain, identified by domain_id, is contained within a parent power capping domain, identified by parent_id. A value of `0xFFFFFFFF` implies that this power capping domain is contained within no parent domains.
uint32 min_cai	The minimum supported power capping averaging interval (CAI), expressed in microseconds. If CPC is supported, this field references CPL0. When min_cai and max_cai fields return the same value, it indicates that CAI is not configurable. This field is set to zero if the platform does not report this value.

uint32 max_cai	The maximum supported power capping averaging interval (CAI), expressed in microseconds. If CPC is supported, this field references CPL0. When min_cai and max_cai fields return the same value, it indicates that CAI is not configurable. This field is set to zero if the platform does not report this value.
uint32 cai_step	The step size between two consecutive CAI supported by this power capping domain. If CPC is supported, this field references CPL0. This field must not be zero if min_cai and max_cai fields report different values.

3.10.3.6 POWERCAP_CPC_ATTRIBUTES

This command returns the CPC attributes associated with a domain. On success, the command returns an array, which contains several CPLi descriptors. Sometimes it might not be possible to return all the descriptors with just one call. To solve this problem, the interface allows multiple calls. It also returns the number of remaining descriptors. The size of the array returned depends on the number of return values a given transport can support. The CPL Identifiers returned by this call should be in numeric ascending order. For an example of using this kind of API, see Section 3.5.6.8.

message_id: `0xD`
protocol_id: `0x18`
This command is mandatory if CPC is supported for any domain.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.
uint32 desc_index	Index of the first CPLi descriptor to be returned in the CPLi descriptor array. Note: This field does not refer to the CPL Identifier.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid domain attributes were returned • NOT_FOUND: if domain_id pertains to a non-existent domain. • OUT_OF_RANGE: if the desc_index is outside of valid range. • NOT_SUPPORTED: if the request is not supported, or the domain does not support CPC. See Section 3.1.4 for more status code definitions.
uint32 num_cpl	**Bits[31:16]** Number of remaining CPLi descriptors. **Bits[15:0]** Number of CPLi descriptors that are returned by this call.
CPLi_DESC desc[N]	An array of CPLi descriptors, of format described in CPLi Descriptor.

CPLi Descriptor

The CPLi_DESC structure describes the properties of each CPL identifier.

uint32 cpli	Integer identifier associated with the concurrent power limit.
uint32 flags	**Bit[31:1]** Reserved, must be zero. **Bit[0]** Power cap configuration support. • If set to 1, this CPLi supports changing power caps. • If set to 0, this CPLi does not support changing power caps.
uint32 min_power_cap	The minimum power value that can be set as the power cap for this CPLi. It is illegal for this field to be zero. When min_power_cap and max_power_cap fields return the same value, it indicates that power cap for this CPLi is not configurable.
uint32 max_power_cap	The maximum power value that can be set as a power cap for this CPLi. It is illegal for this field to be zero. When min_power_cap and max_power_cap fields return the same value, it indicates that power cap is not configurable.
uint32 power_cap_step	The step size between two consecutive power cap values supported by this CPLi. This field must not be zero if min_power_cap and max_power_cap fields report different values.
uint32 min_cai	The minimum supported power capping averaging interval (CAI) for this CPLi, expressed in microseconds. When min_cai and max_cai fields return the same value, it indicates that CAI is not configurable. This field is set to zero if the platform does not report this value.
uint32 max_cai	The maximum supported power capping averaging interval (CAI) for this CPLi, expressed in microseconds. When min_cai and max_cai fields return the same value, it indicates that CAI is not configurable. This field is set to zero if the platform does not report this value.
uint32 cai_step	The step size between two consecutive CAI supported by this CPLi. This field must not be zero if min_cai and max_cai fields report different values.
uint8 name[16]	Null-terminated ASCII string of up to 16 bytes in length describing the CPLi. Table 26 shows the recommended naming convention to be used.

3.10.3.7 POWERCAP_CAP_GET

This command can be used by an agent to get the value of the currently enforced power cap for a domain.

Note

It is possible for the power cap value returned by this command to be stale by the time the command completes, as another power cap change request could have been initiated and completed in the interim.

message_id: `0x4`
protocol_id: `0x18`
This command is mandatory.

Parameters	
Name	**Description**
uint32 domain_id	Identifier for the power capping domain.
uint32 cpli	Concurrent power limit identifier. Must be set to 0 if the domain does not support CPC.

Return values	
Name	**Description**
int32 status	One of, but not limited to, the following: • SUCCESS: if the power cap setting was successfully returned. • NOT_FOUND: if domain_id or cpli are not valid. • NOT_SUPPORTED: if the request is not supported. See Section 3.1.4 for more status code definitions.
uint32 power_cap	The power cap set for the domain and CPLi. A value of 0 indicates that power capping for this domain has been disabled.

3.10.3.8 POWERCAP_CAP_SET

This command allows an agent to specify a power cap on a specific power capping domain to be enforced over the Capping Averaging Interval (CAI).

The actual power consumption of the domain might be different from the power cap value specified by an agent due to the following reasons:

- There is a lower cap set by another agent for the same domain. In this case the platform enforces the lower power cap.
- The domain might be running a workload whose power consumption is below the cap value.
- Transient power oversubscription. A power oversubscription occurs when the power consumption of a domain is above the maximum sustainable power of that domain for a short duration of time without causing system damage or shutdown, while still maintaining the overall power cap over the Power Averaging Interval.

message_id: `0x5`
protocol_id: `0x18`
This command is mandatory if any domain supports configuring the power cap.

Parameters	
Name	**Description**
uint32 domain_id	Identifier for the power capping domain.
uint32 cpli	Concurrent power limit identifier. Must be set to 0 if the domain does not support CPC.

uint32 flags	**Bits[31:2]** Reserved, must be zero. **Bit[1]** Async flag: • Set to 1 if power cap is to be set asynchronously. In this case the call is completed with POWERCAP_CAP_SET_COMPLETE message if bit[0] is set to 0. If bit[0] is set to 1, POWERCAP_CAP_SET_COMPLETE response is not sent. A SUCCESS return code indicates that the platform has successfully queued this command. For more details, see Section 3.10.4.1. • Set 0 to if the power cap is to be set synchronously. In this case, the call will return when the setting has been completed. **Bit[0]** Ignore delayed response: • If the Async flag (bit[1]) is set to 1 when this bit is set to 1, the platform does not send a POWERCAP_CAP_SET_COMPLETE delayed response. • If the Async flag (bit[1]) is set to 1 and this bit is set to 0, the platform sends a POWERCAP_CAP_SET_COMPLETE delayed response. If the Async flag (bit[1]) is set to 0, then this bit field is ignored by the platform.
uint32 power_cap	The power cap to set for the domain. This field should be set to 0 if the agent wishes to disable power capping for the domain. The platform might still choose to apply a power cap for the domain due to requests from other agents or due to platform specific reasons.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the power cap was set successfully for a synchronous request or if the command was successfully enqueued for an asynchronous request. • NOT_FOUND: if the domain_id or cpli are not valid. • NOT_SUPPORTED: if the request is not supported. • INVALID_PARAMETERS: if the requested cap is not supported, or the flags parameter specifies invalid or illegal options. • DENIED: if the calling agent is not allowed to set the power cap for this domain. An example would be if this domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.

3.10.3.9 POWERCAP_MAI_GET

This command can be used by an agent to get the value of the currently enforced power measurement averaging interval for a domain.

—— Note ————

It is possible for the MAI value returned by this command to be stale by the time the command completes, as another MAI change request could have been initiated and completed in the interim.

message_id: 0x6
protocol_id: 0x18
This command is mandatory if any domain supports reading the PAI.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the MAI setting was successfully returned. • NOT_FOUND: if domain_id does not point to a valid domain. • NOT_SUPPORTED: if the request is not supported. See Section 3.1.4 for more status code definitions.
uint32 mai	The power measurement averaging interval set for the domain.

3.10.3.10 POWERCAP_MAI_SET

This command allows an agent to specify the power measurement averaging interval for the power capping domain.

message_id: 0x7
protocol_id: 0x18
This command is mandatory if any domain supports configuring the MAI.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint32 mai	The power measurement averaging interval to set for the domain. This field must not be zero.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the MAI was set successfully. • NOT_FOUND: if the domain identified by domain_id does not exist. • NOT_SUPPORTED: if the request is not supported. • INVALID_PARAMETERS: if the requested MAI is not supported, or the flags parameter specifies invalid or illegal options. • DENIED: if the calling agent is not allowed to set the MAI for this domain. An example would be if this domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.

3.10.3.11 POWERCAP_CAI_GET

This command can be used by an agent to get the value of the currently enforced power capping measurement averaging interval for a domain.

—— **Note** ————

It is possible for the CAI value returned by this command to be stale by the time the command completes, as another CAI change request could have been initiated and completed in the interim.

————————

message_id: `0xE`
protocol_id: `0x18`
This command is mandatory if any domain supports reading the CAI.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.
uint32 cpli	Concurrent power limit identifier. Must be set to 0 if the domain does not support CPC.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the CAI setting was successfully returned. • NOT_FOUND: if domain_id or cpli are not valid. • NOT_SUPPORTED: if the request is not supported. See Section 3.1.4 for more status code definitions.
uint32 cai	The power capping averaging interval set for the domain and cpli.

3.10.3.12 POWERCAP_CAI_SET

This command allows an agent to specify the power capping averaging interval for the power capping domain.

message_id: `0xF`
protocol_id: `0x18`
This command is mandatory if any domain supports configuring the CAI.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint32 cai	The power averaging interval to set for the domain. This field must not be zero.
uint32 cpli	Concurrent power limit identifier. Must be set to 0 if the domain does not support CPC.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if the CAI was set successfully. • NOT_FOUND: if the domain_id or cpli are not valid. • NOT_SUPPORTED: if the request is not supported. • INVALID_PARAMETERS: if the requested CAI is not supported, or the flags parameter specifies invalid or illegal options. • DENIED: if the calling agent is not allowed to set the CAI for this domain. An example would be if this domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.

3.10.3.13 POWERCAP_DOMAIN_NAME_GET

This command returns the extended name associated with a specific power capping domain.

message_id: `0x8`
protocol_id: `0x18`
This command is mandatory only if extended power capping domain name is supported.
Extended power capping domain name is supported when Bit[28] of attributes field of POWERCAP_DOMAIN_ATTRIBUTES command is set to 1. Refer to Section 3.10.3.5 for more details.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid power capping domain extended name is returned. • NOT_FOUND: if domain_id pertains to a non-existent domain. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint8 name[64]	Null-terminated ASCII string of up to 64 bytes in length describing the power capping domain extended name.

3.10.3.14 POWERCAP_MEASUREMENTS_GET

This command allows the calling agent to request the measurement data, like average power consumption and MAI, for the power capping domain.

message_id: `0x9`
protocol_id: `0x18`
This command is mandatory if any domain supports reading power measurements.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the measurements were successfully returned. • NOT_FOUND: if domain_id does not point to a valid domain. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to get the measurements for this domain. An example would be if this domain is exclusive to another agent. See Section 3.1.4 for more status code definitions.
uint32 power	The average power consumption of the power capping domain measured over the latest power measurement averaging interval.
uint32 mai	The power measurement averaging interval set for the domain.

3.10.3.15 POWERCAP_CAP_NOTIFY

This command allows the caller to request notifications from the platform when either power cap or CAI changes for a specific power capping domain. These notifications are sent using the POWERCAP_CAP_CHANGED notification, which is described in Section 3.10.5.1.

Notification support is optional, and PROTOCOL_MESSAGE_ATTRIBUTES must be used to discover whether this command is implemented.

These notifications must be disabled by default during initial boot of the platform.

message_id: `0xA`
protocol_id: `0x18`
This command is mandatory if any domain supports power cap and CAI change notifications.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.
uint32 notify_enable	**Bits[31:1]** Reserved must be zero. **Bit[0]** Notify enable. This bit can have one of the following values: • 0, which indicates that the platform does not send any POWERCAP_CAP_CHANGED messages to the calling agent. • 1, which indicates that the platform sends POWERCAP_CAP_CHANGED messages to the calling agent when the power cap or CAI for the domain is changed. See Section 3.10.5.1 for more details about POWERCAP_CAP_CHANGED notification.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if domain_id does not point to a valid domain. • INVALID_PARAMETERS: if notify_enable specifies values that are either illegal or incorrect. See Section 3.1.4 for more status code definitions.

3.10.3.16 POWERCAP_MEASUREMENTS_NOTIFY

This command allows the caller to request notifications from the platform when the average power consumption of a power capping domain is outside the configured upper or lower power threshold. The average power consumption is measured over the latest power measurement averaging interval. The platform notifies the agent using the POWERCAP_MEASUREMENTS_CHANGED notification, which is described in Section 3.10.5.2.

Notification support is optional, and PROTOCOL_MESSAGE_ATTRIBUTES must be used to discover whether this command is implemented.

These notifications must be disabled by default during initial boot of the platform.

message_id: `0xB`
protocol_id: `0x18`
This command is mandatory if any domain supports power measurements change notification.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain.
uint32 notify_enable	**Bits[31:1]** Reserved must be zero. **Bit[0]** Notify enable. This bit can have one of the following values: • 0, which indicates that the platform does not send any POWERCAP_MEASUREMENTS_CHANGED messages to the calling agent. • 1, which indicates that the platform sends POWERCAP_MEASUREMENTS_CHANGED messages to the calling agent when the power consumption for the domain is outside the band specified by the higher or lower power thresholds. See Section 3.10.5.2 for more details about the POWERCAP_MEASUREMENTS_CHANGED notification.
uint32 power_thresh_low	The lower threshold specified in power units which are in use by this domain. This value replaces any lower threshold previously configured by the calling agent. The platform sends a notification to the agent when the average power consumption of the power capping domain, measured over MAI, falls below this threshold.
uint32 power_thresh_high	The upper threshold specified in power units which are in use by this domain. This value replaces any upper threshold previously configured by the calling agent. The platform sends a notification to the agent when the average power consumption of the power capping domain, measured over MAI, rises above this threshold.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS. • NOT_FOUND: if domain_id does not point to a valid domain. • INVALID_PARAMETERS: if any of the parameters specify values that are either illegal or incorrect. See Section 3.1.4 for more status code definitions.

3.10.3.17 POWERCAP_DESCRIBE_FASTCHANNEL

This command allows the agent to discover the attributes of the FastChannel for the specified power capping domain and the specified message.

The POWERCAP_DOMAIN_ATTRIBUTES command can be used to discover if a power capping domain supports FastChannels. The PROTOCOL_MESSAGE_ATTRIBUTES command can be used to discover if a command, specified by message_id, supports FastChannels.

message_id: `0xC`
protocol_id: `0x13`
This command is optional.

Parameters

Name	Description
uint32 domain_id	Identifier for the power capping domain for which the FastChannel is allocated.
uint32 message_id	Message-id for which the FastChannel is allocated.
uint32 cpli	Concurrent power limit identifier for which the FastChannel is allocated. Must be set to 0 if the domain does not support CPC, or if the message_id does not require cpli as an input parameter.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: If a valid FastChannel is found. • NOT_FOUND: if domain_id does not point to a valid domain or message_id does not point to a valid message, or cpli does not point to a valid Concurrent Power Limit Identifier. • NOT_SUPPORTED: if FastChannel is not supported for this domain or this message. See Section 3.1.4 for more status code definitions.

uint32 attributes	**Bits[31:3]** Reserved. Should be zero in this version of the specification. **Bits[2:1]** Doorbell Register width. This field is only valid if Doorbell Support is set to 1. • If 0, then doorbell register is 8bits wide. • If 1, then doorbell register is 16bits wide. • If 2, then doorbell register is 32bits wide. • If 3, then doorbell register is 64bits wide. **Bit[0]** Doorbell Support. • If 0, then the FastChannel does not have a doorbell register • If 1, then the FastChannel has a doorbell register.
uint32 rate_limit	**Bits[31:20]** Reserved and set to zero. **Bits[19:0]** Rate Limit in microseconds, indicating the minimum time required between successive requests. A value of 0 indicates that this field is not applicable or supported on the platform.
uint32 chan_addr_low	Lower 32 bits of the FastChannel address.
uint32 chan_addr_high	Higher 32 bits of the FastChannel address.
uint32 chan_size	Size of the FastChannel in bytes. The value of this field should be sufficient to accomodate the payload of the message this FastChannel is used for. For more details on payload requirements please refer to Section 3.5.5.1.
uint32 doorbell_addr_low	Lower 32 bits of the doorbell address. This field is not used if doorbell is not supported.
uint32 doorbell_addr_high	Higher 32 bits of the doorbell address. This field is not used if doorbell is not supported.
uint32 doorbell_set_mask_low	Contains a mask of lower 32 bits to set when writing to the doorbell register. If the doorbell register width, n, is less than 32 bits, then only n lower bits are considered from this mask. This field is not used if doorbell is not supported.
uint32 doorbell_set_mask_high	Contains a mask of higher 32 bits to set when writing to the doorbell register. This field is only valid if the doorbell register width is 64 bits. This field is not used if doorbell is not supported.
uint32 doorbell_preserve_mask_low	Contains a mask of lower 32 bits to preserve when writing to the doorbell register. If the doorbell register width, n, is less than 32 bits, then only n lower bits are considered from this mask. This field is not used if doorbell is not supported.
uint32 doorbell_preserve_mask_high	Contains a mask of higher 32 bits to preserve when writing to the doorbell register. This field is only valid if the doorbell register width is 64 bits. This field is not used if doorbell is not supported.

Bits which are set neither in set_mask nor in preserve_mask are to be cleared.

3.10.4 Delayed responses

3.10.4.1 POWERCAP_CAP_SET_COMPLETE

If the agent has changed the power cap asynchronously through POWERCAP_CAP_SET, the platform sends this delayed response to the agent when the power cap setting request has been completed.

message_id: `0x5`
protocol_id: `0x18`
This command is optional if no domain supports asynchronously setting the power cap.

Parameters

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if power cap was set successfully. Other vendor-specific errors can also be generated depending on the implementation. See Section 3.1.4 for more status code definitions.
uint32 domain_id	Identifier for the power capping domain.
uint32 power_cap	The power cap set for the domain.
uint32 cpli	Concurrent power limit identifier. Must be set to 0 if the domain does not support CPC.

3.10.5 Notifications

3.10.5.1 POWERCAP_CAP_CHANGED

If an agent has registered to receive power cap or CAI change notifications for the power capping domain that is identified by domain_id, the platform sends these notifications to that agent when such changes take place.

The platform sends this notification to a subscribing agent when the power cap or CAI of the domain is changed by a different agent or entity in the system, including the platform itself. The platform might autonomously change the power cap of the domain to apply thermal or other constraints. In each of these occurrences, the subscribing agent will be notified so that it can become aware of the change. These notifications are sent after the power cap or CAI transition has completed. Even if either the power cap or the CAI changes, this notification is sent with the latest power cap and CAI settings for the domain.

message_id: `0x0`
protocol_id: `0x18`
This command is optional if no domain supports power cap and CAI change notifications.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent that caused the power cap and CAI change.
uint32 domain_id	Identifier for the power capping domain.
uint32 power_cap	The power cap set for the domain.
uint32 cai	The power capping averaging interval set for the domain.
uint32 cpli	Concurrent power limit identifier for the power capping domain. Must be set to 0 if the domain does not support CPC.

3.10.5.2 POWERCAP_MEASUREMENTS_CHANGED

If an agent has registered to receive power measurement change notifications for the power capping domain that is identified by domain_id, the platform sends these notifications to that agent when such changes take place.

The platform sends this notification to an agent when the average power measurements of the domain, over the MAI, is outside the lower and higher power thresholds specified when registering for the notification.

message_id: `0x1`
protocol_id: `0x18`
This command is optional if no domain supports power measurements change notification.

Parameters

Name	Description
uint32 agent_id	Identifier of the agent that caused this change event. For the current version of the specification, this field is set to 0 to indicate that the platform is the generator of all power consumption change events.
uint32 domain_id	Identifier for the power capping domain.
uint32 power	The average power consumption of the domain when either the upper or the lower power threshold was breached.

3.11 Pin control protocol

This protocol is intended for controlling pins and their configuration. The pin control protocol provides commands to:

- Describe the protocol version.
- Discover implementation attributes.
- List the pins, groups of pins, available functions, and their association with each other.
- Set the parameter configuration and multiplexing of the pins or groups of pins.
- Optionally request exclusive access to a pin or group of pins.
- Optionally configure the permissions of an agent to access a pin or group of pins.

3.11.1 Pin control protocol background

A pin or a pad is a physical interface from a SoC to the outside world. Pins can perform a specific role, for example, supply clocks to a peripheral or transmit data and signal out from a peripheral. Since the number of pins is not as large as the functions required from it, pins are generally multiplexed. This allows configuring a single pin to perform several different functions, though it only performs only one function at a time. A function which requires only a single pin to enable it is referred to as a single-pin function. Some pins can also perform a generic function under the control of the agent. These pins can be used by the agent for implementation defined purposes, for example to transmit a bit pattern. Such pins are also referred to as General Purpose IO (GPIO).

Pins which can be used to enable a specific peripheral or function can be arranged in a group. A group can contain one or more pins. For example, a set of pins which can be used to connect to a UART can form a group. It is illegal for the platform to advertise a mix of individual pins and groups to enable a single-pin function. Functions which are not single-pin can only be enabled through groups.

A group which contains only one pin is referred to as a single-pin group. There can be multiple groups which advertise support for the same function. However, it is illegal to simultaneously select more than one group to enable a function, unless the function provides GPIO capability.

It is possible for a pin to be part of multiple groups, since a pin can serve multiple functions through multiplexing. In this case, it is the responsibility of the agent to ensure that it does not request the platform to set overlapping groups to perform different functions. The platform ensures that it is not possible to set a group of pins to perform a function when one or more pins within that group are already in use for other functions.

Figure 8 shows a simplified example of pins and several groups of pins at a high-level.

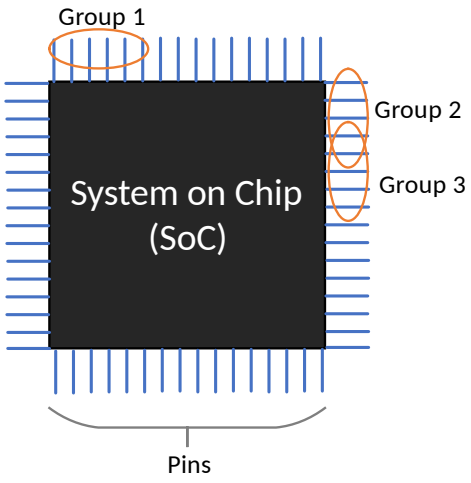

Figure 8: Example of Pins and Groups of Pins

Protocol commands take integer identifiers to describe a pin, group, or function. The identifiers are sequential and start from 0. Though the protocol allows a 32-bit field for identifiers, the upper 16 bits of this field are ignored by the platform. Hence identifiers must be limited to 16 bits.

In a multi-agent system, several scenarios are possible:

- A pin or group is exclusive to an agent.
- A pin or group is shared among multiple agents.

Each agent has their own view of the pin control protocol in terms of attributes and command availability for pins or groups. This view depends on the required agent functionality while maintaining the security, confidentiality, and integrity of other agents in the system. Agents can coordinate with the platform to check and set the function for a pin or group of pins. For all combinations of pins, groups, and agents, platform policy determines which agents can access which pins or groups, and whether a pin or group is shared or exclusive to an agent. It is the responsibility of the platform to resolve requests from different agents for the same pin or group. Platform policy for allocating pins and associated command availability to agents is implementation defined and out of scope of this specification.

A system can include pins or groups that are dedicated exclusively to devices in the Secure or Root address space. These resources are not available for agents in the Non-secure Security state. Such pins or groups must only be managed through Secure or Root channels.

3.11.2 Commands

3.11.2.1 PROTOCOL_VERSION

On success, this command returns the protocol version. For this version of the specification, the return value must be `0x10000`, which corresponds to version 1.0.

message_id: `0x0`
protocol_id: `0x19`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this value must be `0x10000`.

3.11.2.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: `0x10`
protocol_id: `0x19`
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description

uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.11.2.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: `0x1`
protocol_id: `0x19`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 attributes_low	**Bits[31:16]** Number of pin groups. **Bits[15:0]** Number of pins.
uint32 attributes_high	**Bits[31:16]** Reserved, must be zero. **Bits[15:0]** Number of functions.

3.11.2.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

This command can be used to inquire if optional commands are supported, by passing their respective message identifiers to the call. If the platform returns SUCCESS, then it supports the corresponding commands. Otherwise, if the platform returns NOT_FOUND, then it is an indication that the commands are not implemented or not available to the calling agent.

message_id: `0x2`
protocol_id: `0x19`
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is invalid or not implemented. See Section 3.1.4 for more status code definitions.
uint32 attributes	Flags that are associated with a specific command in the protocol. In the current version of the specification, this value is always 0.

3.11.2.5 PINCTRL_ATTRIBUTES

This command returns the attributes associated with a specific function, pin, or a group of pins.

message_id: `0x3`
protocol_id: `0x19`
This command is mandatory.

Parameters

Name	Description
uint32 identifier	Identifier for the pin, group, or function. Identifiers are limited to 16 bits, and the upper 16 bits of this field are ignored by the platform.
uint32 flags	**Bits[31:2]** Reserved, must be zero. **Bits[1:0]** Selector: Whether the identifier field selects a pin, a group, or a function. • 0 – Pin • 1 – Group • 2 – Function All other values are reserved for future use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if valid attributes were returned. • NOT_FOUND: if the identifier field pertains to a non-existent pin, group, or function. See Section 3.1.4 for more status code definitions.

The attributes of the pin, group, or function identified by the identifier field in combination with Bits[1:0] of the flag field in the command.

Bit[31]

Extended name.

- If set to 1, the name is greater than 16 bytes. The extended name is provided by PINCTRL_NAME_GET command which is described in Section 3.11.2.11.
- If set to 0, extended name is not supported.

Bits[30:18]

Reserved, must be zero.

Bit[17]

GPIO function descriptor

- Set to 0 if Bits[1:0] of the flags field in the command is set to 2, and the function does not support GPIO functionality.
- Set to 1 if Bits[1:0] of the flags field in the command is set to 2, and the function supports GPIO functionality.
  The value of this bit must not be 1 for more than one function associated with a pin or a group.

The agent should ignore the value of this bit if Bits[1:0] of flags field in the command is set to 0 or 1.

Bit[16]

Pin-only function descriptor.

- Set to 0 if Bits[1:0] of the flags field in the command is set to 2, and the function is only supported by groups.
- Set to 1 if Bits[1:0] of the flags field in the command is set to 2, the function is a single-pin function, and it is not supported by any group. The function is only supported by individual pins.

The agent should ignore the value of this bit if Bits[1:0] of flags field in the command is set to 0 or 1.

Bits[15:0]

Number of pins or groups.

- Set to 1, if Bits[1:0] of flags field in the command is set to 0.
- Set to the number of pins in the group, if Bits[1:0] of flags field in the command is set to 1.
- Set to the number of pins which can support the function, if Bits[1:0] of flags field in the command is set to 2 and Bit[16] of attributes field is set to 1.
- Set to the number of groups which can support the function in all other cases.

uint8 name[16]	Null-terminated ASCII string of up to 16 bytes in length describing the pin, group, or function name. When Bit[31] of attributes field is set to 1, this field contains the lower 15 bytes of the NULL terminated name. Recommended convention if names are used: • Pins names start with “pin_”, and end with the pin identification in alphanumeric format. An example of a pin name is “pin_x1”. • Group names start with “grp_”, followed by a brief description of the group in alphanumeric format interspersed by ‘_’. An example of a group name is “grp_uart_a_rxtx_1”. • Function names start with “f_”, followed by a brief description of the function in alphanumeric format interspersed by ‘_’. An example of a function name is “f_uart_a”. **NOTE**: General purpose IO (GPIO) function and group names should include 'gpio' in the descriptive literal. An example is “f_gpio”.

3.11.2.6 PINCTRL_LIST_ASSOCIATIONS

This command allows the agent to ascertain:

- the list of pins associated with a group, or
- the list of groups which can enable a function, or
- the list of pins which can enable a single-pin function.

On success, the command returns an array, which contains several pin or group identifiers. Sometimes it might not be possible to return the whole array with just one call. To solve this problem, the interface allows multiple calls. It also returns the number of remaining pins or groups. The size of the array returned depends on the number of return values a given transport can support.

The pin or group identifiers returned by this call should be in numeric ascending order.

For an example of using this kind of API, see Section 3.5.6.8.

A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command. If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

message_id: `0x4`
protocol_id: `0x19`
This command is mandatory only if the number of implemented groups or functions is greater than 0.

Parameters

Name	Description
uint32 identifier	Identifier for the function or group.
uint32 flags	**Bits[31:2]** Reserved, must be zero. **Bits[1:0]** Selector: Whether the identifier field refers to a group or a function. • 0 – Reserved • 1 – Group • 2 – Function All other values are reserved for future use.

uint32 index	Index of the first pin to be described in the return array if the identifier field refers either to a group or to a single-pin function only supported by individual pins. Else it refers to the index of the first group to be described in the return array.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the group or pin identifiers are returned successfully. • NOT_FOUND: if the group or function identified by identifier field does not exist. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to get the pin or group identifiers for this group or function respectively. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:16]** Number of remaining groups or pins. **Bits[15:12]** Reserved, must be zero. **Bits[11:0]** Number of group or pin identifiers that are returned by this call.
uint16 array[N]	Array of group or pin identifiers listed in numerically ascending order. N is specified by Bits[11:0] of flags field.

3.11.2.7 PINCTRL_SETTINGS_GET

This command can be used by an agent to get the pin or group configuration, and the function selected to be enabled. It can also be used to read the value of a pin when it is set to GPIO mode.

The command returns an array of configurations, sorted in numerically increasing config type order. It might not be possible to return all the configuration associated with the pin or group with just one call. To solve this problem, the interface allows multiple calls, with a skip_configs parameter which is used to skip over configurations which were returned by previous calls.

—— Note ———

It is possible for the config value returned by this command to be stale by the time the command completes, as another config set request could have been initiated and completed in the interim.

---

message_id: `0x5`
protocol_id: `0x19`
This command is mandatory.

Parameters

Name	Description
uint32 identifier	Identifier for the pin or group.

uint32 attributes	**Bits[31:20]** Reserved, must be zero. **Bit[19:18]** Config flag • When set to 0, only the configuration value for the configuration type specified by Bits[7:0] needs to be returned. • When set to 1, configuration values for all relevant configuration types associated with the pin or group need to be returned. The returned configuration array is sorted in numerically increasing order of config types. Config types are specified in Table 27. • When set to 2, no configuration values need to be returned. The command only returns the function selected for the pin or the group. All other values are reserved for future use. **Bits[17:16]** Selector: Whether the identifier field refers to a pin or a group. • 0 – Pin • 1 – Group All other values are reserved for future use. **Bits[15:8]** skip_configs The number of configuration types to skip over, before returning the first configuration type and value in the return configuration array. This field is ignored if Bit[19:18] is set to 0 or 2. **Bits[7:0]** ConfigType: The type of configuration whose value is to be returned. Config types are described in Table 27. This field is ignored if Bit[19:18] is set to 1 or 2.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the pin or group configuration and function selection was successfully returned. • NOT_FOUND: if the identifier field does not point to a valid pin or group. • INVALID_PARAMETERS, if the attributes flag specifies unsupported or invalid configurations. • NOT_SUPPORTED: if the request is not supported. • DENIED: if the calling agent is not allowed to get the configuration of this pin or group. See Section 3.1.4 for more status code definitions.
uint32 function_selected	The function currently selected to be enabled by the pin or group specified in the input identifier field. This field is set to `0xFFFFFFFF` if no function is currently enabled by the pin or group specified.

uint32 num_configs	This field should be set to 0 if Bit[19:18] of the attributes field of the command was set to 2. This field should be set to 1 if Bit[19:18] of the attributes field of the command was set to 0. **Bits[31:24]** Number of remaining configurations. **Bits[23:8]** Reserved, must be zero. **Bits[7:0]** Number of configurations that are returned by this call.
{uint32, uint32} configs [N]	Array of configurations, sorted in numerically increasing config type order. This field should be ignored if Bit[19:18] of the attributes field of the command was set to 2. N is specified by Bits[7:0] of the num_configs field. Each array entry is composed of two 32-bit words in the following format: **uint32 entry[0]** type **Bits[31:8]** Reserved, must be zero **Bits[7:0]** config_type The type of config as described in Table 27. **uint32 entry[1]** config_value The configuration value.

Table 27 lists the enumerations and the required pin configuration type.

Table 27: Pin Configuration Type and Enumerations

Enum	Config Type	Values/Units	Enum	Config Type	Values/Units
0	Default	0 - False, 1- True	**12**	Input-mode	0 - Disable, 1- Enable
1	Bias-bus-hold	0 - Disable, 1- Enable	**13**	Pull-mode	0 - Default 1 - Pull up 2 - Pull down 3 – None
2	Bias-disable	0 - Disable, 1- Enable	**14**	Input-value[1]	0 - Low, 1- High
3	Bias-high-impedance	0 - Disable, 1- Enable	**15**	Input-schmitt	0 – Disabled, 1- Enabled
4	Bias-pull-up	Ohms	**16**	Low-power-mode	0 - Disable, 1- Enable
5	Bias-pull-default	0 - Disable, 1- Enable	**17**	Output-mode	0 - Disable, 1- Enable
6	Bias-pull-down	Ohms	**18**	Output-value	0 - Low, 1- High
7	Drive-open-drain	0 - Disable, 1- Enable	**19**	Power-source	Implementation defined units

Enum	Config Type	Values/Units	Enum	Config Type	Values/Units
8	Drive-open-source	0 - Disable, 1- Enable	**20**	Slew-rate	Implementation defined units
9	Drive-push-pull	0 - Disable, 1- Enable	**21 -191**	Reserved	-
10	Drive-strength	Microamperes	**192 -255**	OEM specific units	-
11	Input-debounce	Microseconds			

[1:] Input-value can only be used with PINCTRL_CONFIG_GET. All other usage is illegal.

3.11.2.8 PINCTRL_SETTINGS_CONFIGURE

This command can be used by an agent to set the pin or group configuration, and the function selected to be enabled by the pin or group.

The command allows multiple configurations for the pin or group to be set in a single call. It might not be possible to set all the configuration associated with the pin or group with just one call due to transport limitations. To solve this problem, the agent should make multiple calls sending as many configurations as the transport allows in each call.

message_id: `0x6`
protocol_id: `0x19`
This command is mandatory.

Parameters

Name	Description
uint32 identifier	Identifier for the pin or group.
uint32 function_id	Identifier for the function selected to be enabled for the selected pin or group. This field is set to `0xFFFFFFFF` if no function should be enabled by the pin or group. This value of this field is ignored by the platform if Bit[10] of the attributes field is set to 0.

uint32 attributes	**Bits[31:11]** Reserved, must be zero. **Bit[10]** function_id valid. • When set to 0, the function selection for the pin or group does not need to change. The platform ignores the value of the function_id field. • When set to 1, the function selection for the pin or group needs to change and is specified by the function_id field. **Bits[9:2]** Number of configurations to set. The maximum value of this field is limited by the transport used. The agent needs to specify this field such that the entire command can be accommodated within the transport chosen. This field is set to 0 if no configurations need to be set. If this field is set to 0, Bit[10] must not be 0. **Bits[1:0]** Selector: Whether the identifier field refers to a pin or a group. • 0 – Pin • 1 – Group All other values are reserved for future use.
{uint32, uint32} configs [N]	Array of configs to set, sorted in numerically increasing config type order. N is specified by Bits[9:2] of the attributes field. Each array entry is composed of two 32-bit words in the following format: **uint32 entry[0]** type **Bits[31:8]** Reserved, must be zero **Bits[7:0]** config_type The type of config as described in Table 27. **uint32 entry[1]** config_value The configuration value.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the pin or group configuration, and the function selection was successfully set. • NOT_FOUND: if identifier field does not point to a valid pin or group. • INVALID_PARAMETERS: if the input parameters specify incorrect or illegal values. • NOT_SUPPORTED: if the configuration requested by this command is not supported by the pin or group. • DENIED: if the calling agent is not allowed to set the configuration of this pin or group. • IN_USE: if the pin or group is currently in use by another agent. • PROTOCOL_ERROR: if the platform detects that the number of configurations sent exceeds transport capacity. See Section 3.1.4 for more status code definitions.

3.11.2.9 PINCTRL_REQUEST

This command is used by an agent to request exclusive control of a pin or group. This would make the pin or group unavailable for any other agent until PINCTRL_RELEASE has been called by the controlling agent.

A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command. If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

message_id: 0x7
protocol_id: 0x19
This command is optional.

Parameters

Name	Description
uint32 identifier	Identifier for the pin or group.
uint32 flags	**Bits[31:2]** Reserved, must be zero. **Bits[1:0]** Selector: Whether the identifier field refers to a pin or a group. • 0 – Pin • 1 – Group All other values are reserved for future use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if exclusive control of the pin or group was granted. • NOT_FOUND: if the identifier field does not point to a valid pin or group. • INVALID_PARAMETERS: if the input parameters specify incorrect or illegal values. • DENIED: if the calling agent is not allowed to request this pin or group. • IN_USE: if the pin or group is currently under exclusive control of another agent. See Section 3.1.4 for more status code definitions.

3.11.2.10 PINCTRL_RELEASE

This command is used by an agent to release exclusive control of a pin or group.

A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command. If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

message_id: 0x8
protocol_id: 0x19
This command is optional.

Parameters

Name	Description
uint32 identifier	Identifier for the pin or group.

uint32 flags	**Bits[31:2]** Reserved, must be zero. **Bits[1:0]** Selector: Whether the identifier field refers to a pin or a group. • 0 – Pin • 1 – Group All other values are reserved for future use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the pin or group was successfully released. • NOT_FOUND: if the identifier field does not point to a valid pin or group. • INVALID_PARAMETERS: if the input parameters specify incorrect or illegal values. See Section 3.1.4 for more status code definitions.

3.11.2.11 PINCTRL_NAME_GET

This command returns the extended name associated with a pin, group, or function.

A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command. If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

message_id: `0x9`
protocol_id: `0x19`
This command is mandatory only if extended name is supported.
Extended name is supported when Bit[31] of attributes field of the response to PINCTRL_ATTRIBUTES command is set to 1. Refer to Section 3.11.2.5 for more details.

Parameters

Name	Description
uint32 identifier	Identifier for the pin, group, or function. Identifiers are limited to 16 bits, and the upper 16 bits of this field are ignored by the platform.
uint32 flags	**Bits[31:2]** Reserved, must be zero. **Bits[1:0]** Selector: Whether the identifier field selects a pin, a group, or a function. • 0 – Pin • 1 – Group • 2 – Function All other values are reserved for future use.

Return values

Name	Description

int32 status	One of, but not limited to, the following: • SUCCESS: if valid extended name is returned. • NOT_FOUND: if the identifier field pertains to a non-existent pin, group, or function. See Section 3.1.4 for more status code definitions.
uint32 flags	**Bits[31:0]** Reserved, must be zero.
uint8 name[64]	Null-terminated ASCII string of up to 64 bytes in length describing the extended pin, group, or function name. Recommended naming convention: • Pins names start with “pin_”, and end with the pin identification in alphanumeric format. An example of a pin name is “pin_x1”. • Group names start with “grp_”, followed by a constricted description of the group in alphanumeric format interspersed by ‘_’. An example of a group name is “grp_uart_a_rxtx_1”. • Function names start with “f_”, followed by a constricted description of the function in alphanumeric format interspersed by ‘_’. An example of a function name is “f_uart_a”. **NOTE**: General purpose IO (GPIO) function and group names should include 'gpio' in the descriptive literal. An example is “f_gpio”.

3.11.2.12 PINCTRL_SET_PERMISSIONS

This command is used to indicate to the platform whether an agent has permissions to access a pin or group, as specified by an identifier. An agent can only operate on pins or groups to which it has access. At system boot, the default pin or group specific access permissions of an agent is IMPLEMENTATION defined.

Arm recommends that only trusted agents in the system are permitted to invoke this command.

A caller can discover if this command is implemented by issuing the PROTOCOL_MESSAGE_ATTRIBUTES command and passing the message_id of this command. If the command is implemented, PROTOCOL_MESSAGE_ATTRIBUTES returns SUCCESS.

message_id: `0xA`
protocol_id: `0x19`
This command is optional.

Parameters

Name	Description
uint32 agent_id	Identifier of the Agent.
uint32 identifier	Identifier for the pin or group.

uint32 flags	**Bits[31:3]** Reserved, must be zero. **Bit[2]** Permission: • If set to 0, deny agent access to the pin or group. • If set to 1, allow agent access to the pin or group. **Bits[1:0]** Selector: Whether the identifier field selects a pin or a group • 0 – Pin • 1 – Group All other values are reserved for future use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the permissions were set successfully for the agent specified by agent_id. • NOT_FOUND: if agent_id, or identifier for the pin or group does not exist. • INVALID_PARAMETERS: if flags field is invalid. • NOT_SUPPORTED: if the command is not supported. • DENIED: if the calling agent is not allowed to set the permission for the agent specified by agent_id, or the permissions for the pin or group specified by the identifier. See Section 3.1.4 for more status code definitions.

3.12 System Telemetry protocol

This protocol is intended for gathering system telemetry. The telemetry protocol provides commands to:

- Describe the protocol version.
- Discover implementation attributes.
- List the Telemetry Data Events, sampling rates, and describe the memory mapped Telemetry Data Capture Regions supported.
- Select the Telemetry Data Events and sampling rate to enable.
- Enable or disable telemetry and timestamps.
- Optionally support Telemetry via Continuous Notifications.

3.12.1 Telemetry protocol background

System Telemetry comprises of the process of measuring and subsequently transmitting a range of data which provides insights into the health, usage, performance and functionality of a system. The telemetry protocol allows an agent to request the platform to measure and provide access to a specific set of telemetry data. This data could be used for multiple purposes including:

- long term system monitoring to identify potential issues or faults and to remedy them.
- profiling the system to understand workload and usage characteristics.
- performance tuning or characterization of the system.
- dynamically controlling system features in response to specific phenomena like thermal or power exigencies.

The following are implementation specific and out of the scope of this specification:

- the actual set of data collected by telemetry.
- mechanisms used by the platform for data collection.
- the storage, analysis and visualization of telemetry data.

A key design consideration of telemetry protocol is to minimize agent overhead and to ensure that the system is minimally perturbed during data collection and transmission.

3.12.2 Telemetry protocol concepts

3.12.2.1 Data Event (DE)

Typically, telemetry captures a set of data elements each of which measures a particular aspect of the system. For example, the output of a sensor which measures the temperature of a PE, and the number of times the PE has been powered off because of idleness are two separate data elements.

Each unique data element is identified using a *Data Event (DE)*. Each Data Event is represented by the protocol using a 32-bit unsigned integer identifier. In the example above, the temperature sensor of the PE can be identified using DE identifier 0x0, and the number of times the PE has been powered off because of idleness can be identified using DE identifier 0x1.

The exact description of each DE, and the topological relationship between them, if any, is expected to be captured by an implementation specific mechanism and is out of the scope of this specification.

DEs can optionally be arranged in event groups to ease configuration and control. Each event group is represented by the protocol using a 32-bit unsigned integer identifier starting from 0x0. Event group identifier 0xFFFFFFFF is reserved and must not be used.

The data associated with each DE is expressed using a 64-bit wide sign extended integer field. In the example above, the temperature sensor reading is a 64-bit wide sign extended integer. In case the data exceeds 64-bits in width, multiple entries can be used.

DEs can optionally be accompanied with timestamps that indicate when the associated data was captured. Timestamps should be derived from a single monotonic time base. The selection of a time base is beyond the scope of this specification and should be agreed between the agent and the platform by other standard mechanisms.

3.12.2.2 Telemetry Data Capture Format (TDCF)

Telemetry data is captured by the platform and shared with the agent using the Telemetry Data Capture Format (TDCF).

Each field in the TDCF is 32-bit wide to enable compatibility with systems which only support 32-bit wide (DWORD) data access.

Figure 9 shows the layout of the TDCF.

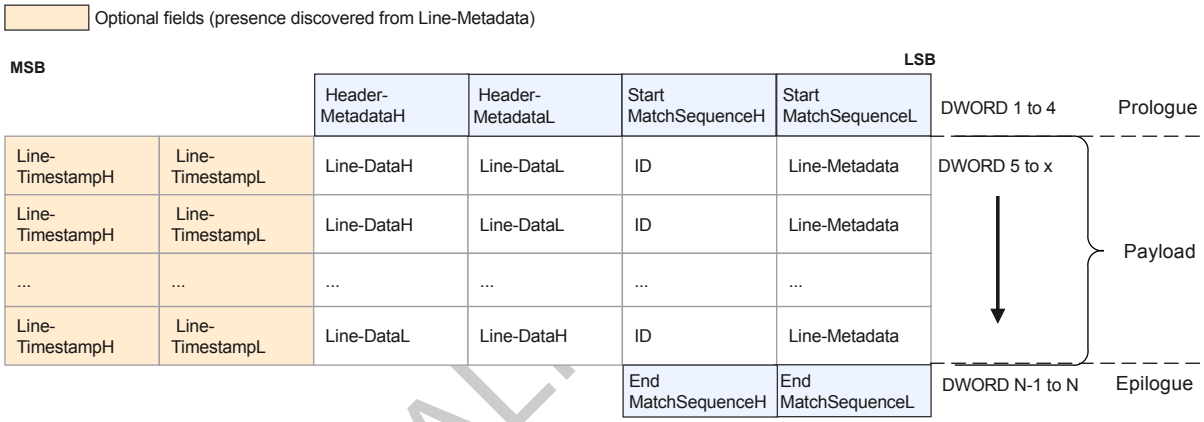

Figure 9: TDCF

The TDCF starts with 16 bytes of Prologue followed by a variable length Payload section. The TDCF is terminated by an 8-byte Epilogue.

Prologue and Epilogue

The Prologue contains Header-Metadata which indicates the number of QWORDS in the Payload section and does not include the length of the Epilogue. A QWORD defined as 64 bits.

The Prologue contains *Start MatchSequence* fields and the Epilogue contains *End MatchSequence* fields. *MatchSequenceH* and *MatchSequenceL* are the respective higher and the lower 32-bits of the MatchSequence.

Race conditions can arise between write accesses by the platform and read accesses by the agent if telemetry is provided via memory regions shared between the agent and the platform. This can lead to stale or invalid values being read by the agent. The MatchSequence is used to detect such race conditions.

If the telemetry is provided via shared memory regions, the MatchSequence is a 4 byte non-zero unsigned value that is populated by the platform and is read by the agent. The MatchSequence should be an even number at boot.

The MatchSequence should be incremented by one, once at the start of, and once at the end of a telemetry update. This ensures that the MatchSequence is an odd number when the Header-Metadata and Payload is being updated and is an even number otherwise.

Care should be taken to handle the case when the MatchSequence wraps to zero. In such a scenario, the platform can choose to reset the MatchSequence to its boot time value.

A flow chart depicting the steps involved when the platform updates telemetry is shown in Figure 10.

The MatchSequence is an even number when telemetry is not being updated. The platform increments the Start and End MatchSequence by one at the beginning of a Header-Metadata and Payload update.

As a result, the MatchSequence becomes an odd number. The platform then updates the Header-Metadata and Payload. When the platform has completed updating the Header-Metadata and Payload sections, it again increments the Start and End MatchSequence by one.

As a result, the MatchSequence becomes an even number when telemetry update is complete. The MatchSequence before and after the update differ from each other. An odd value of the MatchSequence along with a mismatch of values can be used by an agent to detect invalid reads from the shared memory region.

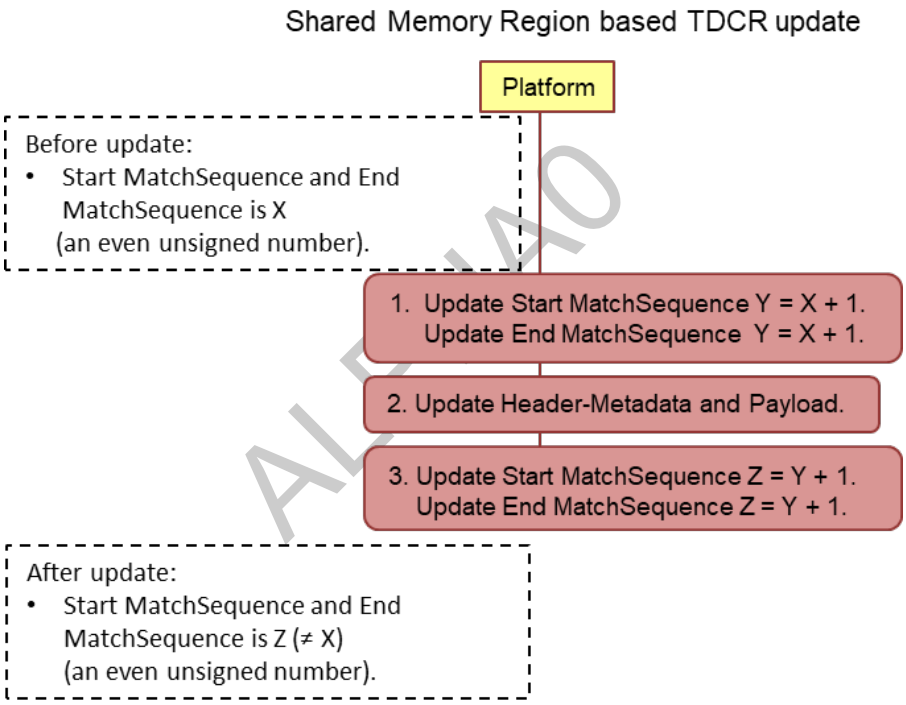

Figure 10: Shared Memory Region based TDCF update by Platform

A flow chart depicting the steps involved when an agent reads the shared memory region is shown in Figure 11.

The agent reads the Start MatchSequence before reading the Header-Metadata and Payload sections. If the value read is odd, the agent aborts the read and retries till the Start MatchSequence read is an even number.

The agent reads the End MatchSequence again at the end of reading the Header-Metadata and Payload. If the Start MatchSequence read before, and the End MatchSequence read after the Header-Metadata and Payload reads are identical, the read was successful.

Otherwise, further reads must be done till the Start MatchSequence and the End MatchSequence are equal, and they are even numbers.

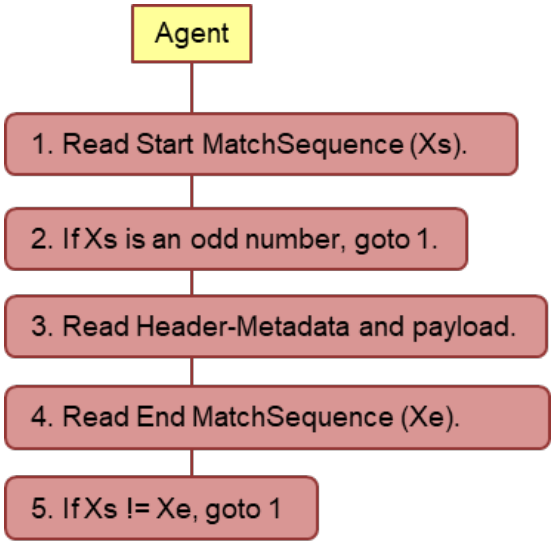

Figure 11: Shared Memory Region based TDCF read by Agent

Payload

The Payload section, as specified in Table 28, is comprised of individual lines, where each line contains the following fields:

- **Line-Metadata**: Indicates additional information about the line entries. This field is 32-bit wide.
- **ID**: Indicates the Data Event ID associated with this line. This field is 32-bit wide.

  Line-Data: Contains the data associated with this line. Line-DataH and Line-DataL are each 32-bit wide. In case the Line-Data exceeds 64-bits, then multiple lines can be used with the same Line-Metadata and ID. Since the payload is represented as a sequence of lines, a later line contains the higher 64-bits of Line-Data.
- **Line-Timestamp**: Contains the timestamp when Line-Data was captured by the platform. Line-TimestampH and Line-TimestampL are each 32-bit wide. The presence of Line-Timestamp is optional, and its presence is indicated by the Line-Metadata field. If the Line-Timestamp fields are not present, then the next line starts immediately after the Line-DataH field.

A line can either be:

- A data line, which contains data associated with a DE identifier.
- A Block Timestamp line, which contains timestamps common to a set of data lines.

The Line-Metadata indicates if the line is a data line or a Block Timestamp line.

Table 28 describes the fields in the TDCF.

Table 28: TDCF fields

Section	Field (DWORD)	Description
Prologue	Start MatchSequenceL	Lower 32-bits of start match sequence.

Section	Field (DWORD)	Description
	Start MatchSequenceH	Higher 32-bits of start match sequence.
	Header-MetadataL	**Bits[31:0]** The number of QWORDS in the Payload section excluding the Epilogue, where QWORD is defined as 64 bits.
	Header-MetadataH	Reserved, must be zero.
Payload Line	Line-Metadata	**Bits[31:5]** Reserved, must be zero. **Bit[4]** Block Timestamp line. • If set to 1, this line contains a Block Timestamp entry, and: – ID must be 0. – Line-DataL constains the lower 32 bits of the Block timestamp. – Line-DataH constains the higher 32 bits of the Block timestamp. – Line-TimestampH and Line-TimestampL fields are not present. – Bits[3:1] must be 0. • If set to 0, this is a data line. **Bit[3]** Uses Block Timestamp. If set to 1, then all the following must be true: • the timestamps associated with this data line are captured by the preceding Block Timestamp line. • Bit[1] must be 1. • Bit[2] and Bit[4] must be zero. If there are multiple Block Timestamp lines preceding this data line, then the timestamp is associated with the Block Timestamp line which is closest to this data line and precedes it. **Bit[2]** Uses Line Timestamp. If set to 1, the timestamps associated with this line are captured by the Line Timestamp fields. In addition, Bit[1] must be 1 and Bits[4:3] must be zero. **Bit[1]** Timestamp valid. • If set to 1, timestamps are associated with this line, and either Bit[2] or Bit[3] must be 1. • If set to 0, no timestamps are associated with this line, and Bits[3:2] must be zero. **Bit[0]** Data invalid. If set to 1, this line contains invalid Line-Data fields. This might be because the platform failed to collect the data requested, the DE was disabled but not deallocated, or because of other reasons. The agent should ignore the entire line.
	ID	Data Event ID.
	Line-DataL	Lower 32-bits of the data.

Section	Field (DWORD)	Description
	Line-DataH	Higher 32-bits of the data.
	Line-TimestampL	Lower 32-bits of the Line Timestamp.
	Line-TimestampH	Higher 32-bits of the Line Timestamp.
Epilogue	End MatchSequenceL	Lower 32-bits of end match sequence.
	End MatchSequenceH	Higher 32-bits of end match sequence.

Figure 12 shows some examples of possible TDCF payload line entries.

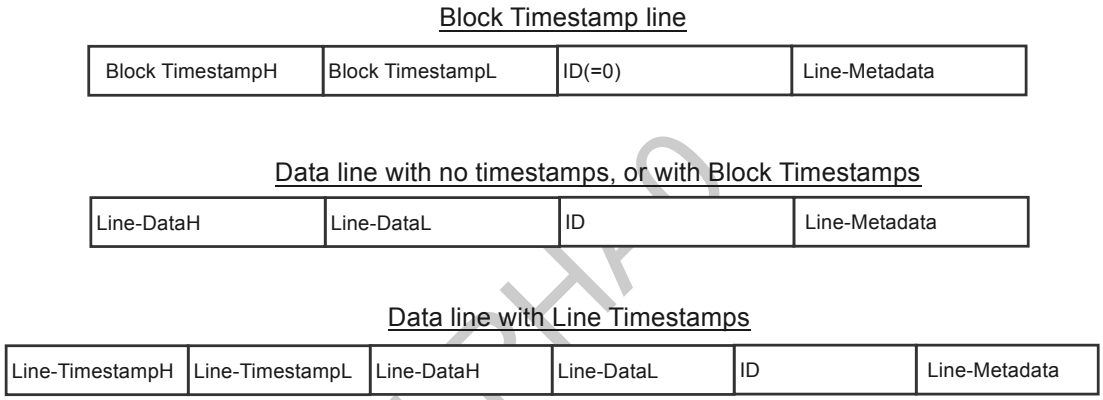

Figure 12: TDCF Payload line examples

3.12.2.3 Telemetry Interfaces

Telemetry can be shared with the agent through any of the following interfaces:

- Shared-Memory based Telemetry Interface (SHMTI), which are memory mapped regions in the agent's physical address space shared between the platform and the agent. Each SHMTI is identified by a unique 32-bit integer identifier which is discoverable using the TELEMETRY_LIST_SHMTI command as specified in Section 3.12.4.5. SHMTIs should be mapped as non-cached device memory and must support the entire TDCF including the Prologue and Epilogue. The Epilogue is located at a fixed position which occupies the last 8 bytes of the shared memory region. DEs which have been enabled and allocated at a particular offset within an SHMTI cannot be relocated by the platform until the DE has been disabled. In any event group, DEs which use SHMTI must be allocated in the same shared memory region.
- SCMI FastChannels, which must guarantee atomicity between reads by an agent and writes by the platform, such that reads always return valid telemetry data. The agent reads the FastChannel whenever it needs to gather telemetry data associated with a specific DE. The FastChannel associated with a DE can be discovered using the TELEMETRY_DE_DESCRIPTION command, as specified in Section 3.12.4.6. The FastChannel size must be sufficient to accommodate the DE data along with timestamps if supported. FastChannel allows a DE to return greater than 64-bits of Line-Data, and must not return a Prologue or Epilogue field. The format of telemetry data captured in a FastChannel is specified in Figure 12, where each DWORD field is 32-bit wide, Line-Data0 is the lowest 32-bit of data, and Line-DataN is the highest 32-bit of data.
- A sequence of DWORDs sent by the platform to the agent over an implementation defined transport.

- Any other mechanism, like FIFOs, queues, and circular buffers, which provides telemetry to the agent using TDCF.

Sharing the same DE data using multiple telemetry interfaces to an agent might result in unpredictable behaviour. For example, when a DE data is available over a FastChannel, it should not be returned using SHMTI.

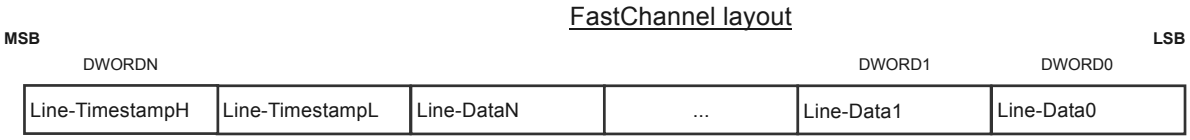

Figure 13: FastChannel Layout

A telemetry interface can omit the Prologue and the Epilogue if it can:

- guarantee atomicity between reads by an agent and writes by the platform such that reads always return valid telemetry data, and
- indicate the number of bytes of payload ready to be read by the agent.

Unless using SCMI FastChannels, all telemetry interfaces must transmit at least a single payload line comprising Line-Metadata, ID, Line-DataH, Line-DataL, and optionally Line-Timestamp if supported.

Discovery of all telemetry interfaces, except SHMTIs and FastChannels, are beyond the scope of this specification.

3.12.3 Telemetry usage by agent

3.12.3.1 Capability discovery and configuration

At boot, telemetry might be enabled or disabled by default by the platform. The platform might nominate special agents which are pre-configured to receive telemetry. The exact mechanism and policy of pre-configuration is platform specific and beyond the scope of this specification.

The agent can use the relevant protocol commands to discover the Data Events, timestamping capabilities and sampling rates available to it.

In a multi-agent system, several scenarios are possible:

- A Data Event is exclusive to an agent.
- A Data Event is accessible by multiple agents.

Each agent has their own view of the telemetry protocol in terms of attributes and command availability. This view depends on the required agent functionality while maintaining the security, confidentiality, and integrity of other agents in the system. For all Data Events and telemetry configuration, platform policy determines which agents can access a Data Event, the supported sampling rates and other configuration. Platform policy for allocating Data Events and associated command availability to agents is implementation defined and out of scope of this specification.

A system can include Data Events that are dedicated exclusively to the Secure or Root address space. These resources are not available for agents in the Non-secure Security state. Such Data Events must only be managed through Secure or Root channels.

—— **Note** ——————————

Telemetry can be exploited for side-channel attacks because they can be used to accurately observe system behavior when sensitive workloads are executing. It is recommended that this protocol is enabled only after due consideration of the potential possibility and mechanism of such attacks. It is also recommended that the Data Event granularity and update interval is restricted on production systems after considering the worst-case data dependent variation of

the system properties which the Data Event is measuring. Additional defense mechanisms might include, but are not restricted to, the introduction of random noise margins to measurements and certificate-based enablement of relevant platform functionality. It is the responsibility of the platform to ensure that this protocol is not exploitable for attacks. The exact mechanism used to identify the deployment parameters of this protocol is beyond the scope of this specification.

3.12.3.2 Telemetry Collection

Protocol commands can be used to determine the physical address and size of the SHMTIs which are memory mapped into the agent's physical address space. Discovery of other types of telemetry interfaces is outside the scope of this specification and must be described by means like firmware table technologies such as FDT or ACPI.

The agent can choose to collect data using continuous update notifications. The agent can also alternatively choose to use the single-sample asynchronous read mode if it just needs to collect a single sample.

The agent enables telemetry collection through the following steps:

1) The agent calculates the number of Data Events it can collect simultaneously according to the properties of the telemetry interface. If the agent exceeds this limit, the platform will return OUT_OF_RANGE error.
2) The agent chooses to either disable and reconfigure all currently enabled Data Events or add to the list of already enabled Data Events.
3) The agent enables the Data Events it wants to collect, indicating if timestamping is required.
4) If not already enabled, the agent enables telemetry collection indicating the sampling rate.

The agent can choose between the following modes of collection:

1) On-demand Interface Read without notifications: Data is read by the agent as required (on-demand) from SHMTIs and FastChannels without any notification from the platform. Data is generated continuously by the platform at the sampling rate configured by the agent. This is the recommended mode of usage when fast sampling rates and low overhead is desired. This mode is only available and must be supported when using SHMTIs and FastChannels.
2) Continuous notification: Data is read by the agent in response to the TELEMETRY_UPDATE notification from the platform as specified in Section 3.12.6.1. The platform generates this notification every time new samples are available, but at an average rate no faster than the sampling rate configured. The notification is used as a trigger by the agent to read telemetry data associated with DEs which are enabled and available over SHMTIs and FastChannels. In addition, this notification includes multiple telemetry payload lines for DEs which are enabled but not available using SHMTI or FastChannels. The agent does not need to explicitly subscribe to the TELEMETRY_UPDATE notification when this mode is used. The platform indicates support for this mode using Bit[30] of attributes_1 field of PROTOCOL_ATTRIBUTES command.
3) Single-sample asynchronous read: This mode provides only a single sample of data to the agent. The platform generates the TELEMETRY_READING_COMPLETE response when the telemetry data sample is available. This response is used as a trigger by the agent to read telemetry data associated with DEs which are enabled and available over SHMTIs and FastChannels. In addition, this response includes multiple telemetry payload lines for DEs which are enabled but not available using SHMTI or FastChannels. Telemetry collection for the agent is automatically disabled once the TELEMETRY_READING_COMPLETE response has been issued by the platform. The platform indicates support for this mode using Bit[31] of attributes_1 field of PROTOCOL_ATTRIBUTES command.

The following table highlights the mechanisms of telemetry collection over different telemetry interfaces.

Table 29: Telemetry Collection Modes

Telemetry Interface Type	Collection Mode		
	On-demand interface read without notifications	**Continuous notification**	**Single-sample asynchronous read**
FastChannel	Supported	Supported (Telemetry Data in Fastchannel)	Supported (Telemetry data in FastChannel)
SHMTI	Supported	Supported (Telemetry Data in SHMTI)	Supported (Telemetry data in SHMTI)
Others	Depends on transport interface	Supported (Telemetry Data in notification payload)	Supported (Telemetry data in delayed response payload)

Note: The platform may support concurrent telemetry collection using multiple telemetry interfaces of different types. For example, some DEs might be collected using SHMTI, while others might be collected over interfaces which only support continuous notification mode of operation.

To enable mixed mode operation, on-demand mode of data collection may be auto-modified by the platform to use continuous notification for telemetry interfaces which do not support on-demand read by agent. As an example, if on-demand mode is set in TELEMETRY_CONFIG_SET command, the collection from SHMTI uses on-demand and the collection from other interfaces may use continuous notification. However, it is illegal for the platform to auto-modify on-demand mode to continuous notification mode of data collection if all the telemetry interfaces support on-demand read by the agent.

When collection is enabled, the platform samples the enabled Data Events at the chosen sampling interval. The platform returns the data to the agent along with the timestamp when the Data Event was collected, if timestamping was enabled.

3.12.4 Commands

3.12.4.1 PROTOCOL_VERSION

On success, this command returns the protocol version. For this version of the specification, the return value must be 0x10000, which corresponds to version 1.0.

message_id: `0x0`
protocol_id: `0x1B`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 version	For this revision of the specification, this value must be `0x10000`.

3.12.4.2 NEGOTIATE_PROTOCOL_VERSION

This command is used to negotiate the protocol version that the agent intends to use, if it does not support the version returned by the PROTOCOL_VERSION command.

There is no limit on the number of negotiations which can be attempted by the agent. All commands, responses, and notifications must comply with the protocol version which was last negotiated successfully. Using protocol versions different from the version returned by PROTOCOL_VERSION without successful negotiation is considered best effort, and functionality is not guaranteed.

message_id: `0x10`
protocol_id: `0x1B`
This command is optional but is strongly recommended to be implemented for platforms that intend to support agents compliant with older versions of this protocol.

Parameters

Name	Description
uint32 version	The negotiated protocol version the agent intends to use.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the negotiated protocol version is supported by the platform. All commands, responses, and notifications post successful return of this command must comply with the negotiated version. • NOT_SUPPORTED: if the protocol version is not supported. See Section 3.1.4 for more status code definitions.

3.12.4.3 PROTOCOL_ATTRIBUTES

This command returns the implementation details associated with this protocol.

message_id: `0x1`
protocol_id: `0x1B`
This command is mandatory.

Return values

Name	Description
int32 status	See Section 3.1.4 for status code definitions.
uint32 de_num	Total number of Data Events supported by this protocol.
uint32 groups_num	Total number of event groups supported by this protocol.

uint32 de_implementation_rev_dword0	Bits[31:0] of the 128-bit DE implementation revision. DE implementation revision is used to indicate changes to Data Events and event groups implemented by the platform. DE implementation revision can be used to indicate changes to DE identifier mappings, DE attribute changes for the same DE identifier, or other changes specific to the way DEs and event groups are interpreted by the agent for a specific platform. The value and interpretation of DE implementation revision is IMPLEMENTATION DEFINED. However, a 128-bit UUID complying to RFC 9562 [UUID] is recommended to be used. All UUID fields should be sequenced in big-endian format in memory.
uint32 de_implementation_rev_dword1	Bits[63:32] of the 128-bit DE implementation revision.
uint32 de_implementation_rev_dword2	Bits[95:64] of the 128-bit DE implementation revision.
uint32 de_implementation_rev_dword3	Bits[127:96] of the 128-bit DE implementation revision.
uint32 attributes_1	**Bit[31]** Single-sample asynchronous read support. • If this flag is set to 1, then the protocol supports single-sample asynchronous telemetry DE reads through the TELEMETRY_CONFIG_SET command. The platform indicates data availability through the TELEMETRY_READING_COMPLETE delayed response. • If this flag is set to 0, this protocol does not support single-sample asynchronous reads. **Bit[30]** Continuous update notification support. • If this flag is set to 1, the protocol supports continuous update notifications. The platform sends the TELEMETRY_UPDATE notification at the configured sampling interval, when new data samples are available over the telemetry interface. • If this flag is set to 0, the protocol does not support continuous update notifications. **Bits[29:19]** Reserved, must be zero. **Bit[18]** Event group specific sampling rate and collection support. • If set to 1, each event group can be configured to a different sampling rate and collection mode using the TELEMETRY_CONFIG_SET command, as specified in Section 3.12.4.10. All DEs which are not a part of any event group must be configured to the same sampling rate and collection mode. • If set to 0, the same sampling rate and collection mode must be configured for all DEs and event groups across the platform using the TELEMETRY_CONFIG_SET command, as specified in Section 3.12.4.10.

	Bit[17] Telemetry Reset support. If set to 1, the platform supports full reset of telemetry infrastructure which includes all configuration and accumulated DE data. **Bit[16]** FastChannel support. • If set to 1, FastChannels are supported. • Set to 0 otherwise. **Bits[15:0]** Number of SHMTIs supported by this protocol.

3.12.4.4 PROTOCOL_MESSAGE_ATTRIBUTES

On success, this command returns the implementation details associated with a specific message in this protocol.

This command can be used to inquire if optional commands are supported, by passing their respective message identifiers to the call. If the platform returns SUCCESS, then it supports the corresponding commands. Otherwise, if the platform returns NOT_FOUND, then it is an indication that the commands are not implemented or not available to the calling agent.

message_id: `0x2`
protocol_id: `0x1B`
This command is mandatory.

Parameters

Name	Description
uint32 message_id	message_id of the message.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: in case the message is implemented and available to use. • NOT_FOUND: if the message identified by message_id is invalid or not implemented. See Section 3.1.4 for more status code definitions.
uint32 attributes	Flags that are associated with a specific command in the protocol. In the current version of the specification, this value is always 0.

3.12.4.5 TELEMETRY_LIST_SHMTI

This command returns the details of the shared-memory based telemetry interfaces (SHMTI) available.

On success, the command returns an array, which contains several SHMTI descriptors. Sometimes it might not be possible to return all the descriptors with just one call. To solve this problem, the interface allows multiple calls. It also returns the number of remaining descriptors. The size of the array returned depends on the number of return values a given transport can support.

For an example of using this kind of API, see Section 3.5.6.8.

message_id: 0x3
protocol_id: 0x1B
This command is mandatory if SHMTIs are supported as indicated by Bits[15:0] of attributes_high field in PROTOCOL_ATTRIBUTES command. Refer to Section Section 3.1.4 for more details.

Parameters

Name	Description
uint32 index	Index of the first SHMTI descriptor to be described in the return array.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if the SHMTI list was returned succesfully. • NOT_FOUND: if no SHMTI is available. • NOT_SUPPORTED: if the request is not supported. See Section 3.1.4 for more status code definitions.
uint32 num_SHMTI	**Bits[31:16]** Number of remaining SHMTIs. **Bits[15:0]** Number of SHMTIs that are returned by this call.
{uint32, uint32, uint32, uint32} SHMTI_desc [N]	SHMTI descriptor. Array of descriptors. N is specified by Bits[15:0] of num_SHMTI field. Each descriptor entry is composed of 4 32-bit words with the following format: **uint32 entry[0]** SHMTI ID. Each SHMTI is identified by a unique 32-bit integer identifier. **uint32 entry[1]** SHMTI address low. The lower 32 bits of the physical address where the SHMTI memory region starts. This value should be 64-bit aligned. The address must be in the memory map of the calling agent. **uint32 entry[2]** SHMTI address high. The higher 32 bits of the physical address where the SHMTI memory region starts. The address must be in the memory map of the calling agent. **uint32 entry[3]** SHMTI length. The length in bytes of the SHMTI region.

3.12.4.6 TELEMETRY_DE_DESCRIPTION

This command can be used for Data Event discovery on the platform. On success, it returns an array of DE Descriptors as described in DE Descriptor.

For an example of using this kind of API, see Section 3.5.6.8.

message_id: 0x4
protocol_id: 0x1B
This command is mandatory.

Parameters

Name	Description

uint32 desc_index	Index of the first DE descriptor to be read in the DE descriptor array.
Return values	
Name	**Description**
int32 status	See Section 3.1.4 for more status code definitions.
uint32 num_desc	**Bits[31:16]** Number of remaining DE descriptors. **Bits[15:0]** Number of DE descriptors that are returned by this current call.
DE_DESC desc[N]	An array of DE descriptors, of format described in DE Descriptor.

DE Descriptor

The DE_DESC structure describes the DE properties, such as the unique identifier for the DE, timestamping support and other characteristics.

uint32 de_id	Identifier for the DE. Each DE is identified by a unique 32-bit integer identifier.
uint32 group_id	Event group identifier for the DE. Set to 0xFFFFFFFF if this DE does not belong to any event group.
uint32 de_data_size	Actual size of the DE data in bytes. The space allocated for the DE data depends on the telemetry interface used, and is a multiple of 8 bytes for SHMTI, and a multiple of 4 bytes for FastChannels.
uint32 de_attributes_1	**Bit[31]** Named specified. • If set to 1, the DE name is specified by the *name* field. • If set to 0, the *name* field is not present. The DE name is not specified. **Bit[30]** FastChannel support. • If this flag is set to 1, the DE values are returned using FastChannels. The fields de_attributes_4, de_attributes_5 and de_attributes_6 are present in this DE descriptor. • If this flag is set to 0, no FastChannel is provisioned for this DE. The fields de_attributes_4, de_attributes_5 and de_attributes_6 are not present in this DE descriptor. The next DE descriptor starts after the de_attributes_3 field. **Bits[29:22]** DE type: The type of Data Event as describe in Table 30. **Bit[21]** Persistent: The DE data values are persistent across all reboot cycles, except cold reboot. In this case the DE ID must not change across reboot cycles where the DE data value persists. **Bits[20:13]** DE unit Exponent: The power-of-10 multiplier in two's complement format that is applied to the unit specified by the DE unit field.

	Bits[12:5] DE unit: The measurement system the DE implements, as described in Table 19. **Bits[4:1]** Timestamp exponent: This field is represented in two's complement format. It is the power-of-10 multiplier that is applied to the DE timestamps (timestamp x $10^{[\text{timestamp exponent}]}$) to represent it in seconds. This field is only valid if Bit[0] is set to 1. **Bit[0]** Timestamp support: • If this flag is set to 1, the DE can provide timestamped values. • If this flag is set to 0, the DE cannot provide timestamped values. Timestamps should be derived from a monotonic time base. The selection of a time base is beyond the scope of this specification and should be agreed between the agent and the platform by other standard mechanisms.
uint32 de_attributes_2	**Bits[31:24]** DE Instance ID: Identifies DEs which measure the same DE unit, as specified by Bits[12:5] of de_attributes_1 field, associated with the same Component Instance. Component Instance is specified by Bits[23:8]. DE instance IDs are unsigned sequential integers starting from 0. For example, if there are 3 DEs within the same CPU which measure temperature at different locations within the CPU, their DE Instance IDs are 0, 1 and 2. This field is set to 0 if there is only one instance of the DE within a component ID. **Bits[23:8]** Component Instance: Instance of the component this DE is associated with. Component Instances are sequential unsigned integers starting from 0. For example if Bits[7:0] is set to 0x0, then this field identifies the CPU instance this DE is associated with. If the component type is a CPU, then this field is recommended to be ordered according to the numeric order of the values read from the architecture specific CPU identifier registers in the system. For example, MPIDR_EL1 is the CPU identifier register in Arm A-Profile CPUs. This field is set to 0 if there is only one instance of the component. **Bits[7:0]** Component Type: The component type this DE is associated with, as described in Table 31.
uint32 de_attributes_3	Reserved, must be zero.
uint32 de_attributes_4	Lower 32 bits of the FastChannel address. This field is not present if Bit[30] of de_attributes_1 field is set to 0.
uint32 de_attributes_5	Higher 32 bits of the FastChannel address. This field is not present if Bit[30] of de_attributes_1 field is set to 0.
uint32 de_attributes_6	Size of the FastChannel in bytes. The FastChannel size must be sufficient to accommodate the DE data along with timestamps if supported. This field is not present if Bit[30] of de_attributes_1 field is set to 0.

uint8 name[16]	A NULL terminated UTF-8 format string with the DE name, of up to 16 bytes. This field is not present if Bit[31] of de_attributes_1 field is set to 0.

Table 30: Data Event Type

DE Type	Description	Interpretation and associated fields
0x0	Unspecified	
0x1	Accumulating - Idle/Sleep State residency	The total time spent in the idle state for the component specified by the Component Type and Component instance fields. The DE Instance ID field is used to identify the idle state type for the component specified by the Component Type and Component instance fields. The idle states are unsigned sequential integer values starting from 0, where 0 indicates active state. Successive higher values indicate increasingly deeper power saving state. It is recommended to align the values according to the idle states as presented to the agent.
0x1	Accumulating - Idle/Sleep State count	The total time count of idle state transitions for the component specified by the Component Type and Component instance fields. The DE Instance ID field is used to identify the idle state type for the component specified by the Component Type and Component instance fields. The idle states are unsigned sequential integer values starting from 0, where 0 indicates active state. Successive higher values indicate increasingly saving state. It is recommended to align the values according to the idle states as presented to the agent.
0x2	Accumulating - others	
	Instantaneous - Idle/Sleep State	The current idle state of the component specified by the Component Type and Component instance fields. The idle states are unsigned sequential integer values starting from 0, where 0 indicates active state. Successive higher values indicate increasingly deeper power saving state. It is recommended to align the values according to the idle states as presented to the agent. The DE Instance ID field is ignored.
0x5	Instantaneous - others	N/A
0x6	Averaging	N/A
0x7	Status	N/A
0x8 - 0xEF	Reserved for future use	N/A

DE Type	Description	Interpretation and associated fields
0xF0 - 0xFF	OEM specified	OEM specified

Table 31 describes DE component types.

Table 31: Data Event Component Type

Component Type	Description	Component Type	Description
0x00	Unspecified	0x10	SoC
0x01	CPU	0x11	System
0x02	Cluster	0x12	Streaming Mode Compute Unit (SMCU)
0x03	GPU	0x13	Accelerator
0x04	NPU	0x14	Battery
0x05	Interconnect	0x15	Battery Charger
0x06	Memory Controller	0x16	PMIC
0x07	L1 Cache	0x17	Board
0x08	L2 Cache	0x18	Memory
0x09	L3 Cache	0x19	Device or Peripheral
0x0A	Last Level Cache	0x1A	Subcomponent (of a device or peripheral)
0x0B	System Cache	0x1B	Lid
0x0C	Display Controller	0x1C	Display
0x0D	Image Processing Unit (Camera)	0x1D -0xDF	Reserved for future use
0x0E	Chiplet	0xE0 - 0xFF	OEM specified
0x0F	Package		

Table 32 describes the DE ID ranges.

Table 32: Data Event ID Reservation

DE ID	Description	Measurement unit
0x0000 – 0x9FFF	Implementation specified	-
0xA000	SoC Energy	microjoules
0xA001	Total CPU energy across all CPUs	microjoules
0xA002	DRAM energy	microjoules
0xA003	GPU energy	microjoules

DE ID	Description	Measurement unit
0xA004	SoC temperature	millidegree C
0xA005	Max. CPU temperature measured across all CPUs	millidegree C
0xA006	GPU temperature	millidegree C
0xA007	DRAM temperature	millidegree C
0xA008	Interconnect frequency	kHz
0xA009	GPU frequency	kHz
0xA00A	Memory bandwidth	Kilobytes per second
0xA00B	DRAM speed	kHz
0xA00C	Number of throttling events - all CPU total	Counts
0xA00D	Number of throttling events - GPU	Counts
0xA00E	Number of throttling events - DRAM	Counts
0xA00F	Number of throttling events - SoC	Counts
0xA010	Throttling active - any CPU	State (binary)
0xA011	Throttling active - GPU	State (binary)
0xA012	Throttling active - DRAM	State (binary)
0xA013	Throttling active - SoC	State (binary)
0xA014 - 0xFFFF	Reserved for future architected ID allocation by this specification.	-
0x10000 - 0xFFFFFFFF	Reserved for future use.	-

3.12.4.7 TELEMETRY_LIST_UPDATE_INTERVALS

This command allows the agent to ascertain the update intervals supported by the protocol. On success, the command returns an array, which contains several update interval entries. Sometimes it might not be possible to return all the update intervals with just one call. To solve this problem, the interface allows multiple calls. It also returns the number of remaining update intervals. The size of the array returned depends on the number of return values a given transport can support.

Sometimes individually describing each update interval might be too onerous. In such cases the command can return only the lowest update interval, the highest update interval, and the step size between two successive update intervals.

The update intervals returned by this call should be in numeric ascending order.

message_id: `0x5`
protocol_id: `0x1B`
This command is mandatory.

Parameters

Name	Description
uint32 index	Index of the first update interval value to be described in the return interval array.
uint32 group_identifier	Event group identifier.

uint32 flags	**Bits[31:4]** Reserved, must be zero. **Bits[3:0]** Selector. • If set to 0, this command applies to all DEs which are not a part of any event group. The command does not apply to any DE which is a part of an event group. The group_identifier field is ignored by the platform. • If set to 1, this command applies to the specified event group only. • If set to 2, this command applies to all DEs and to all event groups. The group_identifier field is ignored by the platform. All other values are reserved for future use. This field must be set to 2 if Bit[18] of attributes_1 field of PROTOCOL_ATTRIBUTES command is set to 0. See Section 3.12.4.3.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if update intervals were returned succesfully. • OUT_OF_RANGE: if the index is outside of valid range. See Section 3.1.4 for status code definitions.
uint32 flags	Descriptor for the update intervals supported. **Bits[31:16]** Number of remaining update intervals. This field should be 0 if Bit[12] is 1. **Bits[15:13]** Reserved, must be zero. **Bit[12]** Return format. • If this bit is set to 1, the Interval Array is a triplet that constitutes a segment in the following form: – interval[0] is the lowest update interval that is supported. – interval[1] is the highest update interval that is supported. – interval[2] is the step size between two successive update intervals. • If this bit is set to 0, each element of the Interval Array represents a discrete sensor update interval. **Bits[11:0]** Number of update intervals that are returned by this call. This field should be 3 if Bit[12] is 1.

uint32 intervals[N]	Interval Array: Each array entry has the following format: **Bits[31:21]** Reserved. **Bits[20:5]** sec – Seconds. **Bits[4:0]** exponent – two's complement format representing the power-of-10 multiplier that is applied to the sec field. The representation is in [sec] x $10^{[exponent]}$ format, in units of seconds. • If Bit[12] of the flags field is set to 0, each array entry is a discrete sensor update interval. • If Bit[12] of the flags field is set to 1, then each entry is a member of the segment {lowest update interval, highest update interval, step size}. N is specified by Bits[11:0] of flags field.

For an example of using this kind of API, see Section 3.5.6.8.

3.12.4.8 TELEMETRY_DE_CONFIGURE

This command enables or disables DEs or event groups. If an event group is specified, then the platform must apply the configuration to either all or none of the DEs in the event group. If the agent requests to enable an already enabled DE, or disable an already disabled DE, the platform returns SUCCESS alongwith the other relevant information.

message_id: `0x6`
protocol_id: `0x1B`
This command is mandatory.

Parameters

Name	Description
uint32 identifier	Identifier for the DE or the event group.
uint32 flags	**Bits[31:4]** Reserved, must be zero. **Bit[3]** Selector. Whether the identifier field refers to a DE or an event group. 0 – DE 1 – Event group **Bit[2]** Disable all DEs or groups. • If set to 1, all DEs or groups should be disabled. The identifier field is ignored by the platform. Any data associated with the DEs or group might be lost. **Bits[1:0]** DE Mode • If set to 0, the DE or group specified by the identifier field is disabled. • If set to 1, the DE or group specified by the identifier field is enabled without timestamps. • If set to 2, the DE or group specified by the identifier field is enabled with timestamps. All other values are reserved for future use. This field must be set to 0, if Bit[2] is set to 1.

Return values	
Name	**Description**
int32 status	One of, but not limited to, the following: • SUCCESS: if the DE or event_group was successfully configured. • INVALID_PARAMETERS: if the input parameters specify incorrect or illegal values. • OUT_OF_RANGE: If the total number of DEs or event groups enabled has reached a maximum limit according to platform capability.
uint32 SHMTI_id	SHMTI ID where the DE or the event group is allocated upon enabling. This field is optional and is set to 0xFFFFFFFF if any of the following are true: • no DE or event group is being enabled. • No SHMTI is supported. • SHMTI is not used for this DE or event group. This could be the case for example, if this DE is associated with a FastChannel. • the platform does not support returning this information.
uint32 TDCF_de_offset	Offset, in bytes, from the beginning of the SHMTI to the start of the DE Line-Metadata. If an event group was specified, then this field refers to the DE in the event group which has the smallest offset within the SHMTI. The agent should ignore this field if the SHMTI_id field is set to 0xFFFFFFFF.

3.12.4.9 TELEMETRY_DE_ENABLED_LIST

This command lists the DEs or groups which have been enabled.

message_id: `0x7`
protocol_id: `0x1B`
This command is mandatory.

Parameters	
Name	**Description**
uint32 index	Index of the first enabled element to be described in the return array.
uint32 flags	**Bits[31:1]** Reserved, must be zero. **Bit[0]** Selector. Whether the index field refers to a DE or an event group. 0 – DE 1 – Event group

Return values	
Name	**Description**
int32 status	One of, but not limited to, the following: • SUCCESS: if update intervals were returned successfully. • OUT_OF_RANGE: if the index specified is outside of valid range.

uint32 flags	**Bits[31:16]** Number of remaining enabled DE array elements. **Bits[15:0]** Number of DE array elements that are returned by this call.
{uint32, uint32} array[N]	Enabled Array. Array of enabled DEs or event groups along with their mode. N is specified by Bits[15:0] of flags field. Each array entry is composed of 2 32-bit words with the following format: **uint32 entry[0]** DE or event group ID **uint32 entry[1]** Bits[31:2] Reserved, must be zero. Bits[1:0] Mode If set to 1, enabled without timestamps. If set to 2, enabled with timestamps. All other values are reserved.

For an example of using this kind of API, see Section 3.5.6.8.

3.12.4.10 TELEMETRY_CONFIG_SET

This command is used to configure and enable telemetry data capture.

message_id: `0x8`
protocol_id: `0x1B`
This command is mandatory.

Parameters

Name	Description
uint32 group_identifier	Event group identifier.
uint32 control	**Bits[31:9]** Reserved, must be zero. **Bits[8:5]** Selector. • If set to 0, this command applies to all DEs which are not a part of any event group. The command does not apply to any DE which is a part of an event group. The group_identifier field is ignored by the platform. • If set to 1, this command applies to the DEs in the specified event group only. • If set to 2, this command applies to all DEs and to all event groups. The group_identifier field is ignored by the platform. All other values are reserved for future use. This field must be set to 2 if Bit[18] of attributes_1 field of PROTOCOL_ATTRIBUTES command is set to 0. See Section 3.12.4.3.

	Bits[4:1] Mode. This field indicates how the agent wants to consume telemetry. It can be set to one of the following modes, depending on the attributes presented to the agent by the PROTOCOL_ATTRIBUTES command, as specified in Section 3.12.4.3. • Set to 0 if the agent intends to read the telemetry data on demand without any notification from the platform. **Note**: The platform ignores this value and uses continuous notification in case the telemetry interface does not support on-demand collection. This auto-conversion is useful if the platform supports concurrent telemetry collection from SHMTI in addition to another interface which does not support on demand mode. However, it is illegal for the platform to auto-modify on-demand mode to continuous notification mode of data collection if all the telemetry interfaces support on-demand read by the agent. • Set to 1 if the agent wants to use continuous notifications. • Set to 2 if this is a single-sample asynchronous read request. Telemetry is automatically disabled by the platform once the TELEMETRY_READING_COMPLETE response has been provided to the agent. • All other values are reserved for future use. This field is ignored by the platform if Bit[0] is set to 0. **Bit[0]** Telemetry enable • Set to 1 to enable telemetry collection. • Set to 0 to disable telemetry collection. Any data associated with the DEs might be lost.
uint32 sampling_rate	The sampling rate to use. **Bits[31:21]** Reserved. **Bits[20:5]** sec – Seconds. **Bits[4:0]** exponent – two's complement format representing the power-of-10 multiplier that is applied to the sec field. The representation is in [sec] x $10^{[exponent]}$ format, in units of seconds. This field is ignored by the platform if any of the following are true: • Bit[0] of the control field is set to 0. • Bits[4:1] of the control field are set to 2.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if telemetry was succesfully enabled. • INVALID_PARAMETERS: if the input parameters specify incorrect or illegal values, or if no DE has been enabled. • OUT_OF_RANGE: If the number of DEs or event groups enabled has reached a maximum limit according to platform or transport capability, especially when using single-sample asynchronous read or continuous notification mode. The agent can either disable a few DEs or event groups or it can disable timestamping and retry.

3.12.4.11 TELEMETRY_CONFIG_GET

This command is used to read back telemetry configuration.

message_id: `0x9`
protocol_id: `0x1B`
This command is mandatory.

Parameters

Name	Description
uint32 group_identifier	Event group identifier.
uint32 flags	**Bits[31:4]** Reserved, must be zero. **Bits[3:0]** Selector. • If set to 0, this command applies to all DEs which are not a part of any event group. The command does not apply to any DE which is a part of an event group. The group_identifier field is ignored by the platform. • If set to 1, this command applies to the DEs in the specified event group only. • If set to 2, this command applies to all DEs and to all event groups. The group_identifier field is ignored by the platform. All other values are reserved for future use. This field must be set to 2 if Bit[18] of attributes_1 field of PROTOCOL_ATTRIBUTES command is set to 0. See Section 3.12.4.3.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if telemetry was succesfully enabled.
uint32 control	**Bits[31:5]** Reserved, must be zero. **Bits[4:1]** Mode. This field indicates how the agent wants to consume telemetry. It can be set to one of the following modes: • Set to 0 if on demand mode is configured rate. • Set to 1 if continuous notifications are enabled. • Set to 2 if a single-sample asynchronous read request is in progress. • All other values are reserved for future use. This field should be ignored by the agent if Bit[0] is set to 0. **Bit[0]** Telemetry enable • Set to 1 if telemetry is enabled. • Set to 0 if telemetry is disabled.

uint32 sampling_rate	The sampling rate to use. **Bits[31:21]** Reserved. **Bits[20:5]** sec – Seconds. **Bits[4:0]** exponent – two's complement format representing the power-of-10 multiplier that is applied to the sec field. The representation is in [sec] x $10^{[exponent]}$ format, in units of seconds. This field is ignored by the platform if any of the following are true: • Bit[0] of the control field is set to 0. • Bits[4:1] of the control field are set to 2.

3.12.4.12 TELEMETRY_RESET

This command is used to reset the entire telemetry infrastructure, including all configuration and accumulated DE data.

message_id: `0xA`
protocol_id: `0x1B`
This command is mandatory if Bit[17] of attributes_1 field in PROTOCOL_ATTRIBUTES command is set to 1.

Parameters

Name	Description
int32 flags	Reserved, must be zero.

Return values

Name	Description
int32 status	One of, but not limited to, the following: • SUCCESS: if telemetry was succesfully reset. • INVALID_PARAMETERS: if the input parameters specify incorrect or illegal values. • DENIED: If the calling agent is not allowed to perform this action.

3.12.5 Delayed Responses

3.12.5.1 TELEMETRY_READING_COMPLETE

This is a delayed response to a TELEMETRY_CONFIG_SET command issued by an agent which requests a single-sample asynchronous read.

This delayed response is used as a trigger by the agent to read telemetry data associated with DEs which are enabled and available over SHMTIs and FastChannels. In addition, this response includes multiple telemetry payload lines for DEs which are enabled but not available using SHMTI or FastChannels.

If the platform determines that there are certain failure conditions in the sensor itself, such as a fault in the DE hardware or related circuitry or logic, it returns HARDWARE_ERROR to report that condition to the caller. The platform returns PARTIAL_ERROR to indicate that some of the telemetry data could not be collected. Other errors apply to the interface itself and are enumerated in Section 3.1.4.

message_id: 0x8
protocol_id: 0x1B
This command is mandatory if the single-sample asynchronous read mode is supported, as specified by Bit[31] of attributes_1 field of PROTOCOL_ATTRIBUTES command.

Return values

Name	Description
int32 status	An appropriate status code, as described in Section 3.1.4. In case of PARTIAL_ERROR, some data-lines, as indicated by Bit[0] of Line-Metadata, might still be valid and therefore consumable.
uint32 num_dwords	The size of the returned array specified in DWORDS, where DWORD is defined as 4 bytes. The value specified by this field must not be an odd number. This field is set to 0 if all the enabled DEs are collected using SHMTI or FastChannels.
uint32 array[N]	The returned telemetry payload section. The prologue and epilogue are not provided. N is specified by num_dwords field.

3.12.6 Notifications

3.12.6.1 TELEMETRY_UPDATE

The TELEMETRY_UPDATE notification is sent to an agent which had requested continuous update notifications using the TELEMETRY_CONFIG_SET command. The platform might also nominate special agents which receive this notification by default since they were auto configured to receive telemetry. The exact mechanism and policy of autoconfiguration is beyond the scope of this specification.

This notification is used as a trigger by the agent to read telemetry data associated with DEs which are enabled and available over SHMTIs and FastChannels. In addition, this notification includes multiple telemetry payload lines for DEs which are enabled but not available using SHMTI or FastChannels.

The notification is generated at an average rate no faster than the configured sampling period. The platform is allowed to generate the notification at a rate slower than the configured update period if the agent or the platform cannot cope with the rate of generation of notifications due to hardware constraints. However, the platform should always provide the latest DE values when the TELEMETRY_UPDATE notification is generated. The status field can be used to indicate error conditions resulting in non-availability or corruption of telemetry samples. When the platform determines that there are certain failure conditions in the sensor itself, such as a fault in the DE hardware or related circuitry or logic, it returns HARDWARE_ERROR to report that condition to the caller. The platform returns PARTIAL_ERROR to indicate that some of the telemetry data could not be collected.

message_id: 0x0
protocol_id: 0x1B
This notification is mandatory if the platform supports continuous update notifications, as specified by Bit[30] of attributes_1 field of PROTOCOL_ATTRIBUTES command.

Return values

Name	Description
uint32 agent_id	Refers to the agent that caused this event. For the current version of the specification, this field is set to 0 to indicate that the platform is the generator of all telemetry notification events.

int32 status	An appropriate status code, as described in Section 3.1.4. If this field is non-zero, the agent is recommended to ignore the rest of the fields in this notification and re-configure telemetry. However in case of PARTIAL_ERROR, some data-lines, as indicated by Bit[0] of Line-Metadata, might still be valid and therefore consumable.
uint32 num_dwords	The size of the returned array specified in DWORDS, where DWORD is defined as 4 bytes. The value specified by this field must not be an odd number. This field is set to 0 if all the enabled DEs are collected using SHMTI or FastChannels.
uint32 array[N]	The returned telemetry payload section. The prologue and epilogue are not provided. N is specified by num_dwords field.

4 Transports

Transport describes how messages are exchanged between agents and the platform. The transport should allow the agent and the platform to exchange the largest possible message described in this specification. If the platform supports messages which return data elements iteratively, like SENSOR_LIST_UPDATE_INTERVALS as described in Section 3.7.2.7, the transport should allow at least one data element to be returned.

4.1 Shared memory based transport

This form of transport relies on the use of shared memory between the platform and the agents.

The transport optionally supports interrupt-based communication, where, on completion of the processing of a message, the caller receives an interrupt. Polling for completion is also supported.

The transport can be used to provide an agent to platform, or a platform to agent channel. Each channel in the transport includes:

- **Shared memory area**
  This is an area of memory that is shared between the caller and the callee. At any point in time, the shared memory is owned by the caller or the callee. The ownership is reflected by a **channel status** word in the shared memory area. The channel is said to be free when the memory area is owned by the caller, and busy when it is passed to the callee. When a channel is free, the caller can write a message and the associated payload to this shared memory area. After this, the caller updates the status field, thereby relinquishing ownership of the shared memory and marking the channel as busy. The callee can then use the shared memory to pass return values that are associated with the processing of the message. When the callee has completed processing the message, it updates the channel status field to indicate that the channel is now free. The layout of the memory area is described in Section 4.1.2.
- **Doorbell**
  This is a mechanism that the caller can use to alert the callee of the presence of a message.

  Typically, this mechanism is implemented as a register in caller, which, when written, raises an interrupt on the callee. In case the callee chooses to poll over the ‘Channel free’ bit in the Channel status field of the shared memory area to discover new messages from the caller, then the doorbell support is optional.

  The doorbell can also be implemented through Secure Monitor Call (SMC) or Hypervisor Call (HVC) instructions if the callee is resident in the Secure or Root world or at a different exception level.
- **Completion interrupts**
  This transport supports polling or interrupt driven modes of communication. In interrupt mode, when the callee completes processing a message, it raises an interrupt to the caller. Hardware support for completion interrupts is optional.

4.1.1 Message communications flow

A flow chart for sending a message from the caller to the callee using interrupt mode is shown in Figure 14. The steps are equally applicable to commands where the agent is the caller, and delayed responses and notifications where the platform is the caller. The steps are as follows:

1. The caller must ensure that the channel is free.
2. The caller populates the shared memory area with the message and its payload.
3. The caller marks the channel as busy by updating the channel status.
4. The caller rings the doorbell. This signals the callee that a pending message is in the shared memory area.
5. The callee processes the command in shared memory area.

6. Optionally, the callee updates the shared memory area with any return data that are associated with the message processing.
7. The callee marks the channel as free by updating the channel status.
8. The callee issues a completion interrupt to the caller.
9. Optionally the caller processes the contents of the shared memory area.

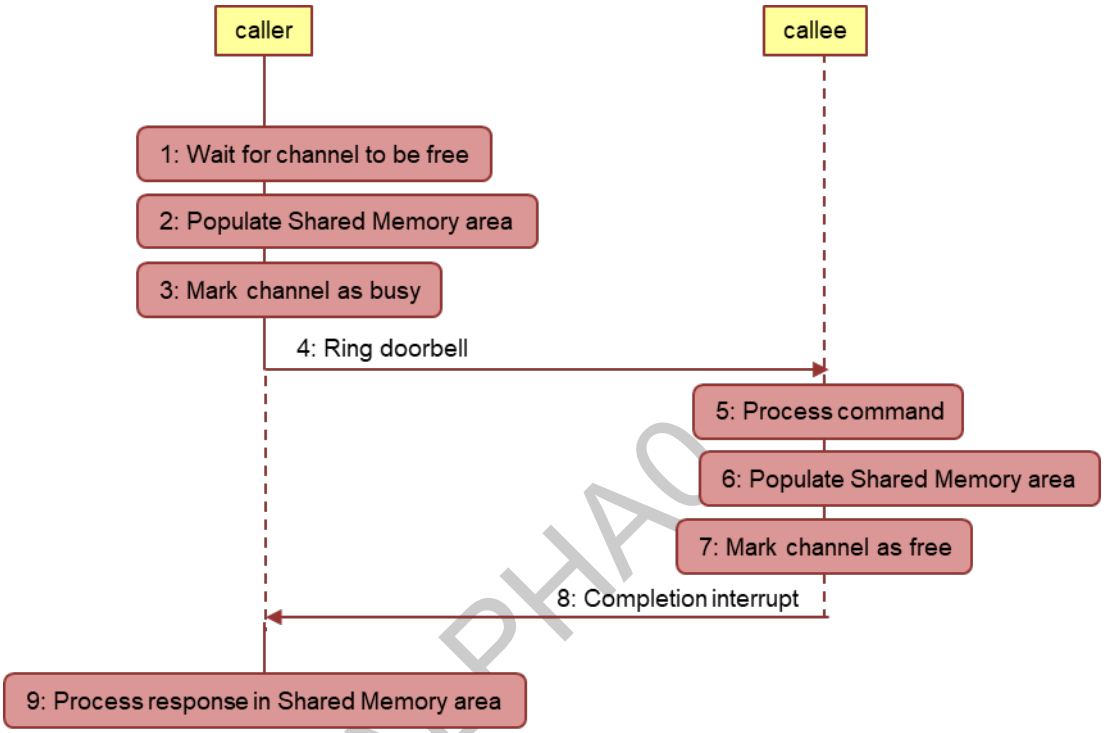

Figure 14: Interrupt-driven Communications flow

A flow chart for sending a message using polling mode is shown in Figure 15. The main difference is that the caller must poll for command completion by checking the status of the channel, as there is no completion interrupt.

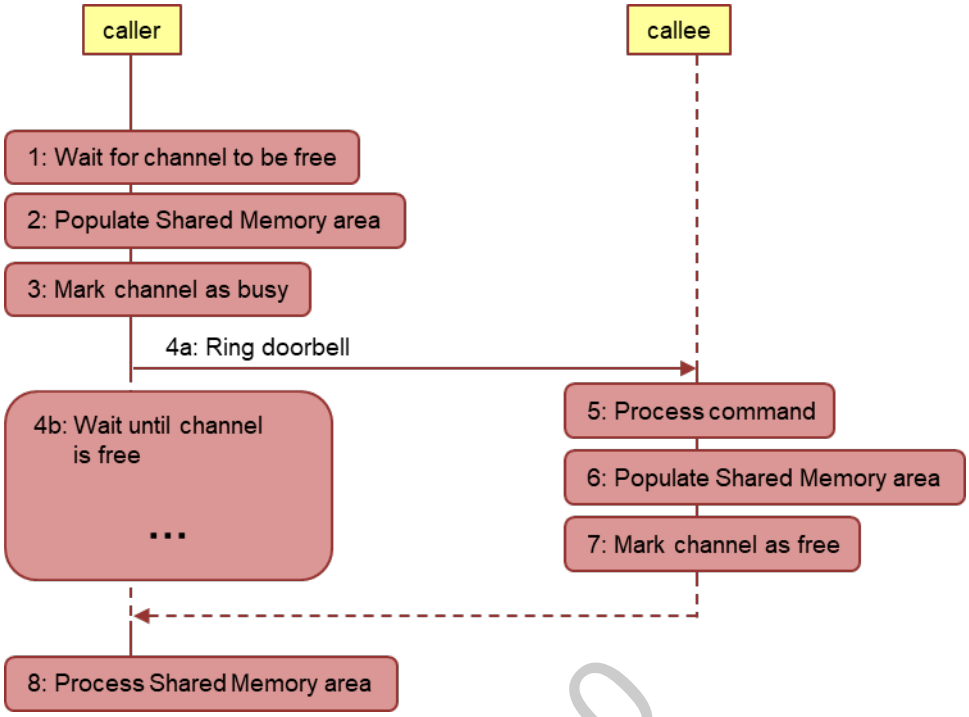

Figure 15: Polling based Communication Flow

The caller must ensure the appropriate ordering of memory operations so that all updates to the shared memory must be visible to the callee before ringing the doorbell. Equally, the callee must ensure that all shared memory changes are visible to the caller before updating the status.

If the caller contains multiple processing elements that can share a transport channel, then appropriate locking must be put in place to ensure that only one processing element can use the channel at any one time. The channel must be locked until the message processing completes and the results are processed by the caller.

4.1.2 Shared memory area layout

For a given channel, the layout of the memory that is shared between the agent and platform is described in Table 33.

Table 33: Layout of the shared memory area

Field	Byte Length	Byte Offset	Description
Reserved	0x4	0x0	Reserved, must be zero.

Field	Byte Length	Byte Offset	Description
Channel status	0x4	0x4	The field has the following format: **Bits[31:2]** Reserved, must be zero. **Bit[1]** Channel error This bit is set to 1 if the previous message was not transmitted due to a communications error. The caller must clear it when it has ownership of the channel. **Bit[0]** Channel free • This bit is set to 1 if the channel is free. • This bit is cleared to 0 if the channel is busy.
Reserved	0x8	0x8	IMPLEMENTATION DEFINED field.
Channel flags	0x4	0x10	Channel flags are described in Table 34.
Length	0x4	0x14	Total Length in bytes of the Message header and the command payload (4+N). If the message length does not match the message, the payload must contain the PROTOCOL_ERROR status as the first return value upon completion of message processing. The length is updated to the total size of the return message header and the return payload upon completion of message processing. Status codes are described in detail in Section 3.1.4.
Message header	0x4	0x18	Message header field as described in Table 3.
Message Payload	N	0x1C	Array of 32-bit values that are used to hold any parameters or return values. The arguments are sent out in the same order they are declared in a protocol command. Return values are sent back in the same order as they are declared in a protocol command. If a message is not known to the callee, the payload must contain NOT_SUPPORTED as the first return value. Status codes are described in detail in Section 3.1.4.

When interrupt driven communication is supported, the transport allows the caller to choose between interrupt and polling driven communications. This can be done on any transfer and is useful when the caller wants to operate in a fire and forget fashion, without having to handle interrupts. To make the choice, the channel flags are used. The format of the flags is described in Table 34.

Table 34: Channel flags

Field	Description
Bits[31:1]	Reserved, must be zero.
Bit[0]	Interrupt communication enable: Set to 1 if the command should complete via an interrupt. Set to 0 if the command should not result in an interrupt assertion.

4.1.3 Firmware representation guidelines

An operating system on an agent needs a description of the shared memory based transport and its properties before using it. Arm recommends using firmware technologies such as FDT and ACPI for this purpose. This section details the properties that are required to be defined for each channel.

4.1.3.1 Doorbell

For agent to platform channels, a doorbell is required to alert the platform that a message is present in the shared memory area. In case the doorbell is a register, writing to it requires a read-modify-write sequence. Firmware tables can be used to describe the properties of the register to an OSPM running on the AP. The properties that must be described are shown in Table 35.

Table 35: Properties of the doorbell register

Field	Description
Register address	Physical address of the register that is written to, to issue a command to the platform.
Preserve Mask	Mask of bits that must be preserved when modifying the doorbell register to issue a command.
Modify Mask	Mask of bits that must be set when modifying the doorbell register to issue a command.

Channels can share a register address for the doorbell, but in this case must have unique preserve and modify masks. If the callee chooses to poll over the 'Channel free' bit in the Channel status field of the shared memory area to discover new messages from the caller, then doorbell support is optional.

If the doorbell is SMC or HVC based, it should follow the SMC Calling Convention [SMCCC]. The doorbell needs to provide the identifier of the Shared Memory area that contains the payload. The Shared Memory area containing the payload is updated with the SCMI return response when the call returns.

For platform to agent channels, a message interrupt can be described. This interrupt is raised by the platform on notification or delayed response messages. Not describing this interrupt implies that that platform messages must be polled by agents.

4.1.3.2 Shared memory area address and size

The physical address of the shared memory area, and its size, must be described to the OSPM.

4.1.3.3 Completion interrupt

For agent-to-platform channels where interrupt mode is supported, the properties of the completion interrupt, if present, must be described by agent firmware. The properties of the completion interrupt to be described are covered in Table 36.

Table 36: Properties of the completion interrupt

Field	Description
Interrupt identifier	Identifier for the interrupt asserted by the platform on command completion.
Interrupt properties	Whether interrupt is level or edge triggered.
Register address	If the interrupt is level sensitive, the physical address of the interrupt clearing register that must be written to, to clear the interrupt.

Field	Description
Preserve Mask	If the interrupt is level sensitive, mask of bits that must be preserved when accessing the register to clear the interrupt.
Modify Mask	If the interrupt is level sensitive, mask of bits that must be set when accessing the register to clear the interrupt.

If the interrupt is level-sensitive, it can be shared by more than one channel. In this case, the preserve-and modify-masks must be unique for each channel.

4.2 ACPI-based Transport

ACPI-based implementations can leverage SCMI protocols to provide platform services using standard ACPI methods. For example, a device may be power managed by an ACPI-aware OS using the standard ACPI control methods that are described in [ACPI]. These ACPI methods can send SCMI Power Management Protocol requests to the platform to transition the power state of the device. In such an implementation, the platform is an ACPI-compliant platform controller as defined by Chapter 14 of [ACPI]. As an example, SCMI transport channels can be represented as an ACPI Platform Communications Channel (PCC) of Type 3. SCMI transports that follow the format outlined in Section 4.1 are compatible with PCC type 3 channel definition. Also, ACPI version 6.3 introduces the concept and use of PCC operation regions. This enables ACPI methods that rely on underlying SCMI services to access the SCMI transport through PCC operation regions.

SCMI FastChannels can be represented as ACPI System Memory and used in the Continuous Performance Control (CPC) object when using ACPI Collaborative Processor Performance Control (CPPC). It is not possible to utilise CPPC unless SCMI FastChannels are used. If Level Indexing Mode is used for a Performance Domain, it is not possible to utilise ACPI CPPC for that domain.

4.3 Shared Memory or MMIO based Transport for FastChannels

FastChannels might rely on the use of shared memory between the platform and the agents. Alternatively, FastChannels can be MMIO based. Any MMIO or shared memory based FastChannel must be visible and readable by both the caller and the callee. However, only the caller or the callee, but not both, must have write permissions to enforce unidirectionality. FastChannels must be mapped as non-cached device memory.

A FastChannel:

a) must be the same width as the payload requirements of the message for which the FastChannel is used. The payload layout of the FastChannel is described in the relevant Protocol sections.

b) can have optional doorbell support. The doorbell can be used to inform the platform that the agent has posted a new request over the FastChannel. If doorbell support is absent, the platform might need to poll over the FastChannel for any messages from the agent.

The discovery of the FastChannel is described in the relevant Protocol sections.

4.4 Virtio-SCMI

The VIRTIO specification enables guests in a virtual machine to use Virtio as SCMI transport using a Virtio SCMI Device. For more details refer to [VIRTIO].