



Realm Management Monitor specification

Document number	DEN0137
Document quality	ALP
Document version	1.1-alp11
Document confidentiality	Non-confidential
Document build information	29b3d2ce doctool 0.56.1

Copyright © 2022-2024 Arm Limited or its affiliates. All rights reserved.

DRAFT

Realm Management Monitor specification (Draft)

This document is a DRAFT of the Realm Management Monitor specification.
Please send feedback to the Arm CCA architecture team.

Realm Management Monitor specification

DRAFT

Release information

1.1-alp11 (10-12-2024)

New features

None

Clarifications

- Clarify the operation performed by the RemExtend function (FENIMORE-902)
- Clarify that the RecAuxCount function returns an IMPLEMENTATION_DEFINED value (FENIMORE-907)
- Rename RsiRdevValidateIoFlags to RsiDevMemFlags (FENIMORE-908)
- RMI_VDEV_COMMUNICATE: explain why PDEV pointer is provided as input value (FENIMORE-910)
- RmmEntryAt function: change signature (FENIMORE-911)
- {RMI,RSI}_VERSION: use “supported” rather than “implemented” (FENIMORE-913)
- Rename RmmEntry::s2ap_{base,overlay} to s2ap_{bidx,oidx} (FENIMORE-912)
- Change type of RmmEntry::s2ap_{bidx,oidx} to UInt4 (FENIMORE-912)
- Change RmmRealmFlags1::rtt_s2ap_indirect to rtt_s2ap_encoding (FENIMORE-912)
- Make consistent statements regarding stage 2 permission fault at Unprotected IPA (FENIMORE-917)
- RMI_DATA_CREATE: add missing arc in diagram from RIPAS=DESTROYED, when Realm state is NEW (FENIMORE-906)
- Clarify return to Realm of VDEV instance ID, in flow diagram (FENIMORE-936)
- Clarify Host behaviour in “Realm shared memory protocol” (FENIMORE-926)
- RSI_RDEV_VALIDATE_MAPPING: make arguments in flows consistent with command definition (FENIMORE-927)
- Clarify attributes stored in REC during device mapping validation flow (FENIMORE-928)
- Clarify description of MEC policy
- Clarify that SPDm and IDE may be optional for devices which are protected “by system construction” (FENIMORE-934)
- RMI_DATA_CREATE: add success condition regarding Granule contents (FENIMORE-935)
- RSI_HOST_CALL: add data.gprs to footprint (FENIMORE-937)
- RSI_HOST_CALL: remove rec.pending from footprint (FENIMORE-937)
- Clarify device communication flows, with and without SPDm (FENIMORE-942)
- Add concept of Granule category (FENIMORE-953)
- Remove PaIsDelegable() checks from NS accesses (FENIMORE-956)
- REC exit due to VDEV communication: constrain rec_exit.exit_reason (FENIMORE-962)
- Add description and references for “(Non-)TEE memory” and “T=1” (FENIMORE-963)
- RmiVsmmuParamsIsValid: clarify criteria for aidr, idr value to be valid (FENIMORE-964)
- Clarify wording regarding the set of GICv3 List Registers which are available (FENIMORE-967)
- RsiFeatureRegisterEncode: add RmmRealm argument (FENIMORE-971)

Defects

- RMI_RTT_SET_S2AP: add success condition which updates rec.s2ap_addr (FENIMORE-817)
- Formalise RttEntriesInRangeRipas function (FENIMORE-911)
- RMI_RTT_SET_RIPAS: formalise da_pre failure condition (FENIMORE-911)
- Model S2AP direct / indirect encoding explicitly (FENIMORE-912)
- RTT folding: memory attributes are only valid if HIPAS is ASSIGNED_{DEV,NS} (FENIMORE-916)
- Specify that an RTT entry is live if its state is VSMMU (FENIMORE-914)
- RMI_RTT_READ_ENTRY: add ASSIGNED_DEV, ASSIGNED_VSMMU (FENIMORE-915)
- Enable Realm to determine whether an device memory IPA range maps to a Limited Order Region (FENIMORE-783)
- Enforce XN=1 for MMIO locations (FENIMORE-819, reverting change made in 1.1-alp10)
- RMI_VSMMU_MAP: remove redundant failure condition on upper IPA bound (FENIMORE-938)
- If MAX_MECID is not zero, default state of all MECs is PRIVATE_UNASSIGNED (FENIMORE-930)
- RsiDeviceInfo: add attestation_type (FENIMORE-941)
- RMI_VSMMU_MAP: correct granule state pre-condition check (FENIMORE-939)
- RMI_RTT_DESTROY: correct state_unprot post-condition (FENIMORE-940)
- RMI_DEV_MEM_MAP: add success conditions for memory attributes (FENIMORE-924)
- RMI_RTT_READ_ENTRY: add support for VSMMU (FENIMORE-965)

- Remove “REC exit due to RTT request” (FENIMORE-945)
- RmiRecExitReason: re-encode to remove gaps
- VDEV: add VDEV_NEW state (FENIMORE-952)
- RMI_DEV_MEM_UNMAP: add “level” input value (FENIMORE-960)
- RMI_RTT_DEV_MEM_VALIDATE: add checks for memory attributes and output address (FENIMORE-929)

Relaxations

None

1.1-alp10 (31-10-2024)

New features

- Virtual SMMU (FENIMORE-887)
- Update quality of “DA foundation” to BETA

Clarifications

- Rename RdevFromIds to RdevFromInstId (FENIMORE-884)
- Clarify definition of device identifier
- Re-order participants in DA flow diagrams to match those in other flows
- Clarify failure modes of VDEV request flow (FENIMORE-860)
- RmiFeatureRegister0: rename PLANE_RTT to RTT_PLANE (FENIMORE-875)
- RmiRealmFlags1: rename rtt_tree_pp to rtt_tree_per_plane (FENIMORE-875)
- RmiRealmFlags1::rtt_tree_per_plane is ignored if the Realm has no auxiliary Planes (FENIMORE-875)
- Replace “device certificate” with “device certificate chain” (FENIMORE-879)
- MEC: change statements regarding change of encryption context from rule to info (FENIMORE-880)
- RTT functions: pass RmmRealm instead of RD address
- PDEV setup flow: consistently number steps in diagram and text (FENIMORE-882)
- PDEV setup flow: remove SPDM_GET_DIGESTS call (FENIMORE-868)
- RMI_RTT_DESTROY: make “summary of command dependencies” table consistent with command (FENIMORE-892)
- Clarify description of output address of an ASSIGNED_NS RTT entry (FENIMORE-895)
- Permit context variables to depend on output values
 - Required to make the “rtte” context variable well-defined in RMI_RTT_READ_ENTRY (FENIMORE-896)
- RTT creation flow: remove constraint on Realm state (FENIMORE-897)
- RMI_DATA_DESTROY: make “output values” table consistent with success conditions (FENIMORE-899)
- Device certificate chain digest does not include an SPDM header (FENIMORE-867)
- Rename RDEV_NEW to RDEV_UNLOCKED (FENIMORE-891)
- {RMI,RSI_VERSION}: formalise failure and success conditions (FENIMORE-898)

Defects

- RMI_VDEV_DESTROY: decrement pdev.num_vdevs (FENIMORE-885)
- RMI_PDEV_DESTROY: fail if num_vdevs is not zero (FENIMORE-885)
- Add number of VDEVs to criteria for Realm liveness (FENIMORE-885)
- RMI_PDEV_SET_PUBKEY: add metadata (FENIMORE-878)
- RMI_PDEV_ABORT: allow aborted transaction to be restarted (FENIMORE-881)
- Remove support for shared MMIO (FENIMORE-883)
- RSI_RDEV_GET_MEASUREMENTS: simplify ABI by following SPDM GET_MEASUREMENTS (FENIMORE-890)
- RMI_RTT_DESTROY: add post-conditions for Unprotected IPA (FENIMORE-893)
- RMI_RTT_READ_ENTRY: add success condition for walk_level output (FENIMORE-896)
- RSI_REALM_CONFIG: revert accidental change to FID (FENIMORE-900)
- RMI_RTT_INIT_RIPAS: check state of all entries in range (FENIMORE-894)
- Add “S2AP indirect encoding” feature (FENIMORE-875)
- RSI_RDEV_GET_INFO: remove public key digest (FENIMORE-750)
- RSI access to a Protected IPA observes HIPAS and RIPAS values (FENIMORE-760, FENIMORE-862)
- RMI_RTT_AUX_MAP_PROTECTED: require primary RTTE to be valid (FENIMORE-870)
- RMI_DEV_MEM_MAP: add level input value (FENIMORE-886)

- Add RSI_RDEV_GET_INSTANCE_ID (FENIMORE-901)
- Prevent cacheable mappings to MMIO locations (FENIMORE-888)
- Do not enforce XN=1 for MMIO locations (FENIMORE-819)

Relaxations

None

1.1-alp9 (30-09-2024)

New features

None

Clarifications

- Clarify rules regarding execution of RSI_HOST_CALL by Pn
- State the device functionality required by the RMM (FENIMORE-837)
- Clarify transfer of GIC ownership on Plane exit

Defects

- RMI_REC_{CREATE,DESTROY}: add realm.num_recs to footprint (FENIMORE-859)
- RMI_RTT_SET_S2AP: don't return RMI_ERROR_RTT_{_AUX} if S2AP overlay index already matches requested value (FENIMORE-869)
- Remove RmiPdevFlags::spdm (FENIMORE-872)
- Simplify PDEV attributes and creation flags (FENIMORE-871)
- RMI_VDEV_ABORT: transition to VDEV_ERROR state (FENIMORE-856)
- RMI_RTT_AUX_{CREATE/DESTROY/FOLD}: return RMI_ERROR_RTT_AUX, not RMI_ERROR_RTT (FENIMORE-863)
- Update interrupt and timer rules to accommodate Pn being GIC owner (FENIMORE-866)
- Fix defects in HIPAS / RIPAS transition diagrams (FENIMORE-857)
 - The only permitted transition to RIPAS=IO is from HIPAS=ASSIGNED_DEV_*, RIPAS=EMPTY.
- RMI_RTT_AUX_UNMAP_{PROTECTED,UNPROTECTED}: return RMI_ERROR_RTT_AUX, not RMI_ERROR_RTT (FENIMORE-873)

Relaxations

None

1.0-relo (10-09-2024)

Clarifications

- RMI_RTT_READ_ENTRY: add ripas_prot success condition
- Clarify rules regarding Realm EL1 timer state
- Correct wording in “Initialize memory of New Realm” flow
- RecAuxCount return value is not greater than 16, and constant for a Realm (FENIMORE-796)
- Clarify purpose of CCA platform hash algorithm ID claim (FENIMORE-811)
- Clarify behaviour of RMI_REC_ENTER if RMI_EMULATED_MMIO flag is set following a REC exit not due to emulatable Data Abort
- RMI_RTT_READ_ENTRY: simplify expression of ripas_unprot pre-condition (FENIMORE-847)

Defects

- Correct typo in “REC entry” section [I_LFYDV]
- Add rule regarding Realm execution of data cache invalidate by set / way (FENIMORE-734)
- Remove SH from the set of Host-controlled Unprotected RTT attributes (FENIMORE-736)
- If LPA2 is enabled, ensure that PA written to RTTE is less than 2^{48} (FENIMORE-752)
- RMI_RTT_SET_RIPAS: if base address is not aligned with entry at which RTT walk terminates, only fail if RIPAS of that entry does not match the requested value (FENIMORE-765)
- RMI_DATA_DESTROY: if address is mapped as block, level can be either 1 or 2 (FENIMORE-775)
- RMI_RTT_MAP_UNPROTECTED: remove reference to non-existent output value “nl” (FENIMORE-776)

- Make number of GICv3 List Register values discoverable (FENIMORE-779)
- RMI_REC_ENTER: if RMI_INJECT_SEA is set then RMI_EMULATED_MMIO is ignored (FENIMORE-782)
- Impose IMPLEMENTATION_DEFINED limit on maximum number of RECs per Realm (FENIMORE-800)
- Allow Realm to query RIPAS of an IPA range (FENIMORE-802)
- Introduce RIPAS DEV value (FENIMORE-802)
- Add RPV to RsiRealmConfig (FENIMORE-810)
- Expand RmiFeatureRegister0::{NUM_BPS, NUM_WPS} to support up to 64 counters (FENIMORE-759)
- Attestation token: change profile value to be a versioned tag (FENIMORE-809)
- RSI_ATTESTATION_TOKEN_CONTINUE: add RSI_ERROR_UNKNOWN failure condition (FENIMORE-832)
- RmiFeatureRegister0::GICV3_NUM_LRS: report number of available LRs, minus one (FENIMORE-845)
- Simplify definition of NUM_BPS, NUM_WPS fields (FENIMORE-846)
- RMI_RTT_READ_ENTRY: ripas_unprot failure condition: change && to || (FENIMORE-861)
- RMI_RTT_INIT_RIPAS: correct inconsistency between text and command definition (FENIMORE-864)
- Fix defects in HIPAS / RIPAS transition diagrams (FENIMORE-857)
 - For transitions due to execution of RMI_RTT_DESTROY, remove from the diagram and describe in text.

Relaxations

- RMI_RTT_{INIT,SET}_RIPAS: relax “top_rtt_align” failure condition
 - The previous condition caused the command to fail if the “top” address was misaligned
 - This is replaced with “no_progress”, which only fails if the command does not modify any RTT entries

1.1-alp8 (29-08-2024)

New features

- Allow P0 to transfer GIC ownership to Pn on Plane entry (FENIMORE-855)

Clarifications

- REC exit from Pn: amend “HIPAS DESTROYED” to “RIPAS destroyed” (FENIMORE-825)
- Rename RMI_PDEV_SET_KEY -> RMI_PDEV_SET_PUBKEY (FENIMORE-824)
- Amend description of AUX_DESTROYED state (FENIMORE-822)
- Rename RsiIoCoherent -> RsiDevMemCoherent (FENIMORE-829)
- Clarify behaviour of RMI_REC_ENTER if RMI_EMULATED_MMIO flag is set following a REC exit not due to emulatable Data Abort
- RMI_RTT_READ_ENTRY: simplify expression of ripas_unprot pre-condition (FENIMORE-847)
- Reduce usage of term “IO” in DA-related parts of the spec (FENIMORE-854)
 - RmiIoAction -> RmiVdevAction
 - RmiIoData -> RmiDevCommData
 - RmiIoDelegateFlags -> RmiDevDelegateFlags
 - RmiIoEnter -> RmiDevCommEnter
 - RmiIoEnterStatus -> RmiDevCommStatus
 - RmiIoExit -> RmiDevCommExit
 - RmiIoExit -> RmiDevCommExitFlags
 - RmiIoRequestType -> RmiDevCommProtocol
 - RmiIoShared -> RmiDevMemShared
 - RmmIoState -> RmmDevCommState
 - RmmIoShared -> RmmDevMemShared
 - REC exit due to IO -> REC exit due to device communication
 - RmiRecExit::io_vdev -> RmiRecExit::vdev
 - RmiRecExit::io_action -> RmiRecExit::vdev_action
 - RIPAS IO -> RIPAS DEV
 - Granule states IO_{PRIVATE,SHARED}, IO_DELEGATED_{PRIVATE,SHARED}, IO_UNDELEGATED -> DEV_{PRIVATE,SHARED}, DEV_DELEGATED_{PRIVATE,SHARED}, DEV_UNDELEGATED
 - RMI_GRANULE_IO_{DELEGATE,UNDELEGATE} -> RMI_GRANULE_DEV_{DELEGATE,UNDELEGATE}
 - RMI_IO_{CREATE,DESTROY} -> RMI_DEV_MEM_{MAP,UNMAP}
 - RSI_RDEV_VALIDATE_IO -> RSI_RDEV_VALIDATE_MAPPING

- Replace “Plane exit due to IRQ” with “Plane exit due to synchronous exception”
- Update ECDSA reference (FENIMORE-831)

Defects

- RMI_RTT_AUX_DESTROYED: on success, change RTTE state to AUX_DESTROYED (FENIMORE-827)
- RSI_PLANE_ENTER: fail if plane_idx is zero (FENIMORE-826)
- Add RsiRealmConfig::gicv3_vtr (FENIMORE-807)
- On REC entry, go to P0 if Pn has a maintenance interrupt pending (FENIMORE-830)
- RSI_ATTESTATION_TOKEN_CONTINUE: add RSI_ERROR_UNKNOWN failure condition (FENIMORE-832)
- Remove RMI_RTT_AUX_READ_ENTRY (FENIMORE-823)
- RMI_RTT{ _AUX }DESTROY sets S2AP overlay index to 0 (FENIMORE-820)
- RMI_RTT_AUX_MAP_{ PROTECTED, UNPROTECTED }: correct calculation of output address (FENIMORE-828)
- RmiFeatureRegister0::GICV3_NUM_LRS: report number of available LRs, minus one (FENIMORE-845)
- Simplify definition of NUM_BPS, NUM_WPS fields (FENIMORE-846)
- Add RMI_{ PDEV, VDEV }_AUX_COUNT; remove RmiFeatureRegister0::{ PDEV, VDEV }_NUM_AUX (FENIMORE-833)
- Allow RMI_FEATURES to report that auxiliary RTTs are not supported (FENIMORE-842)
- Replace IDE_{ LINK, SEL, LINK_SEL } with “device protection” enum (FENIMORE-848)
- Enforce that vdev.tdi_id is unique within a segment (FENIMORE-849)
- Enforce that vdev.tdi_id is within PDEV RID range (FENIMORE-850)
- Enforce 1:1 mapping from vdev_id to VDEV within a Realm (FENIMORE-851)
- Add RMI_VDEV arguments required for locking discipline (FENIMORE-852)
- Cacheability attributes for device memory are defined by stage 1 (FENIMORE-858)

1.1-alp7 (09-07-2024)

New features

- Memory Encryption Contexts (MEC) (FENIMORE-788)
- Support for coherent devices and platform devices (FENIMORE-814)
 - Introduce concepts of platform-attested and independently-attested devices
 - Describe differences and commonalities in communication flows for devices with / without SPD
 - RMI_PDEV_CREATE: replace device class with flags (SPDM, IDE)
 - PDEV: increase maximum number of auxiliary granules to 32
 - VDEV: introduce auxiliary granules
 - RMI_IO_CREATE
 - * Remove flags input value
 - * Permit MemAttr to specify either a cacheable or a non-cacheable mapping
 - Rename RSI_RDEV_CONFIG -> RSI_RDEV_GET_INFO
 - RSI_RDEV_GET_INFO: permit execution in any RDEV state
 - Introduce REC exit due to VDEV request, RMI_VDEV_COMPLETE
 - RSI_RDEV_VALIDATE_IO: add “coherent” flag

Clarifications

- Fix typo in “HIPAS change while Realm state is ACTIVE” diagram (FENIMORE-794)
- Correct command signatures in “Realm validation of MMIO” flow (FENIMORE-795)
- Rename Firmware Component Measurement Log (FCML) -> Firmware Activity Log (FAL) (FENIMORE-798)
- Correct wording in “Initialize memory of New Realm” flow
- RecAuxCount return value is not greater than 16, and constant for a Realm (FENIMORE-796)
- Clarify behavior of Pn instruction fetch from Unprotected IPA
- Fix pseudocode for S2AP change (FENIMORE-805)
- Clarify purpose of CCA platform hash algorithm ID claim (FENIMORE-811)
- Clarify that RSI_RDEV_VALIDATE_IO does not permit change from RIPAS DESTROYED (FENIMORE-785)

Defects

- Rename RSI_RDEV_GET_DIGESTS -> RSI_RDEV_CONFIG (FENIMORE-751)
- Allow Host to select PDEV hash algorithm (FENIMORE-751)

- Impose IMPLEMENTATION DEFINED limit on maximum number of RECs per Realm (FENIMORE-800)
- Remove RmiIoRequestType::RMI_DISCOVERY (FENIMORE-772)
- RMI_RTT_SET_S2AP: add rtt_tree output value (FENIMORE-806)
- Allow Realm to query RIPAS of an IPA range (FENIMORE-802)
- Allow RSI_HOST_CALL execution by Pn to directly exit to the Host (FENIMORE-804)
- On REC exit from Pn, timer state reported to Host is the earliest of P0 and Pn timers (FENIMORE-801)
- Move Pn EL1 sysreg access from RSI_PLANE_ENTER to RSI_PLANE_REG_{READ,WRITE} (FENIMORE-773)
- Add RPV to RsiRealmConfig (FENIMORE-810)
- Expand RmiFeatureRegister0::{NUM_BPS, NUM_WPS} to support up to 64 counters (FENIMORE-759)
- Add S2AP to criteria for RTT homogeneity, when HIPAS=ASSIGNED (FENIMORE-818)

Relaxations

None

1.1-alp6 (31-05-2024)

New features

- Introduce Live Firmware Activation policy (FENIMORE-780)
- Add flag in platform token, indicating whether a software component may be live-activated (FENIMORE-781)
- Make number of GICv3 List Register values discoverable (FENIMORE-779)
- Support for countersignatures on software components (FENIMORE-793)

Clarifications

- Fix typo in “HIPAS change while Realm state is NEW” diagram
- Add RTT_AUX to Granule state diagram (FENIMORE-784)
- Planes: clarifications
 - Exception model
 - * Amend initial state in “Between a Plane entry and a Plane exit, a REC exit and REC entry may occur”
 - * Remove statement that “On Plane entry, all RsiPlaneEnter fields are ignored unless specified otherwise”
 - * Regarding “An exception due to any of the following in Pn cause a REC exit to the Host ...”
 - Introduce “REC exit from Pn” heading to provide appropriate context
 - Separate synchronous and asynchronous exceptions
 - * Add rule that REC entry with a Pending virtual interrupt causes return to P0
 - * Reword info statements regarding action taken by P0 in response to a Plane exit, to remove suggestion that this is an exhaustive list
 - * Remove statement that “all other RsiPlaneExit fields are zero” following Plane exit due to Synchronous Exception
 - Memory management
 - * Reword overview to introduce “S2AP base index”
 - * Remove (“The number of S2AP overlay indices is 16”) as this is specified by the Arm ARM
 - * Reword description of execute permission in Unprotected IPA space
 - * RmiRealmFlags: remove comment that rtt_tree_pp is ignored if the Realm has a single Plane
 - * Correct implementation note to say that S2AP cookie should always be zero if the Realm is configured to use a shared RTT tree
 - * Remove incorrect statement that RSI_MEM_SET_PERM_INDEX does not return a “top” address
- DA
 - Fix typo in RDEV state diagram
 - Make explicit the relationship between RDEV state and SMMU enablement (FENIMORE-787)
 - Remove SPDM_CHALLENGE step from PDEV setup flow (FENIMORE-761)

Defects

- RSI_MEM_SET_PERM_VALUE: fail if Plane index is zero (FENIMORE-778)
- RMI_REC_ENTER: if RMI_INJECT_SEA is set then RMI_EMULATED_MMIO is ignored (FENIMORE-782)
- RMI_PDEV_COMMUNICATE: add missing success conditions (FENIMORE-786)
- Separate Realm flags into measured and unmeasured sets (FENIMORE-792)
 - Rename RmiRealmFlags -> RmiRealmFlags0

- Move `rtt_tree_pp` from `RmiRealmFlags0` to `RmiRealmFlags1`

Relaxations

None

1.1-alp5 (30-04-2024)

New features

None

Clarifications

None

Defects

- Planes: add rules regarding EL1 timers
- `RMI_VDEV_{COMMUNICATE,CREATE}`: add `pdev.num_vdevs` to footprint (FENIMORE-774)
- `RMI_DATA_DESTROY`: if address is mapped as block, level can be either 1 or 2 (FENIMORE-775)
- `RMI_RTT_MAP_UNPROTECTED`: remove reference to non-existent output value “nl” (FENIMORE-776)
- Reduce number of available memory permission overlay indices to 15 (FENIMORE-777)

Relaxations

None

1.1-alp4 (03-04-2024)

New features

None

Clarifications

- Add rationale for permitting `RSI_RDEV_GET_MEASUREMENTS` while the `RDEV` is in `RDEV_NEW` state (FENIMORE-745)
- Add rationale for Host storage of device attestation evidence (FENIMORE-741)

Defects

- `RmmRdev`: add linkage to corresponding `RmmVdev` (FENIMORE-746)
- `RMI_PDEV_SET_PUBKEY`: add failure condition for invalid key length (FENIMORE-740)
- `RMI_RTT_SET_RIPAS`: if base address is not aligned with entry at which RTT walk terminates, only fail if RIPAS of that entry does not match the requested value (FENIMORE-765)
- Specify that maximum number of auxiliary Planes is either 0 or 3 (FENIMORE-769)
- Specify S2AP change flow (FENIMORE-770)
 - Follow a pattern similar to the RIPAS change flow, including allowing the Host to reject an S2AP change request.
- Remove misleading transition to `PDEV_ERROR` in response to Host reporting `RMI_IO_ERROR` (FENIMORE-764)
- Rename `RMI_SECURE_SPDM` to `RMI_SECURE_CMA_SPDM` (FENIMORE-767)
- `RMI_PDEV_ABORT`: amend failure / success conditions to match state transition diagram (FENIMORE-768)
- `RMI_PDEV_SET_PUBKEY`: add success condition which sets `io_state` to `IO_PENDING` (FENIMORE-771)

Relaxations

None

1.1-alp3 (31-01-2024)

Clarifications

- Update “RTT walk” section to reflect addition of auxiliary RTTs (FENIMORE-737)
- `RmiPdevEventIsValid()`: remove `pdev` argument (FENIMORE-747)
- `RMI_RTT_DESTROY`: add ordering “`rtte_state < rtt_live`” (FENIMORE-748)
- Planes: fix typo in description of memory permission overlay index lock bit (FENIMORE-753)

- VDEV teardown flow: remove RMI_IO_DESTROY calls (FENIMORE-756)
- Clarify meaning of “secure SPDM” (FENIMORE-743)
- Address review feedback on Planes chapter
 - Replace use of undefined term “P0 memory” (in reference to RsiPlaneRun object) with “Realm memory”.
 - Clarify that the access permissions which can vary between Planes are at stage 2.
 - Clarify the subset of REC exits in which an RTT tree index is returned to the Host.
 - Use “rec_exit” and “plane_exit” to avoid ambiguity.
 - Append to list of Pn actions which result in REC exit to the Host.
 - Each Plane has a unique VMID, regardless of whether there exists an RTT tree per Plane.
 - Limit the statement “if an IPA is auxiliary-live then the corresponding entry in the primary RTT is live” to apply only to Protected IPA space.
 - Correct statement regarding variance of stage 2 permissions among Planes for an Unprotected IPA.

Defects

- If LPA2 is enabled, ensure that PA written to RTTE is less than 2^{48} (FENIMORE-752)
- On REC exit due to Data Abort / Instruction Abort, provide the index of the RTT tree (FENIMORE-749)
- RSI_MEM_SET_PERM_INDEX: add success condition which sets lock bit (FENIMORE-754)
- On RIPAS change to RAM, reset memory permission overlay index to 0 (FENIMORE-755)
- Introduce RMI_ERROR_RTT_AUX (FENIMORE-757)
- RSI_PLANE_ENTER: add plane_idx; move run_ptr from output to input (FENIMORE-758)
- RSI_PLANE_ENTER: add failure and success conditions
- Add RMI_PDEV_IDE_RESET command (FENIMORE-726)
- Reassign FIDs for commands added in RMM 1.1, to avoid overlap between RMI and RSI

Relaxations

None

1.1-alp2 (11-12-2023)

New features

None

Defects

- RSI_RDEV_VALIDATE_IO: add “private / shared” flag (FENIMORE-732)
- Add rule regarding Realm execution of data cache invalidate by set / way (FENIMORE-734)

Relaxations

None

Defects

- RMI_RTT_READ_ENTRY: add success condition for HIPAS=ASSIGNED_IO_* (FENIMORE-733)
- Remove SH from the set of Host-controlled Unprotected RTT attributes (FENIMORE-736)

1.1-alp1 (06-11-2023)

New features

- Planes (FENIMORE-731)
 - RmiFeatureRegister0: add RTT_TREE_SINGLE, MAX_NUM_AUX_PLANES fields
 - RmiRealmFlags: add RTT_TREE_PP field
 - RmiRealmParams: add members
 - * num_aux_planes
 - * rtt_tree_pp
 - * aux_vmid
 - * aux_rtt_base
 - RmmRealm
 - * Add members

- rtt_tree_pp
 - num_aux_planes
- * Change type of members to “array of per-Plane values”
 - vmid
 - rtt_base
- Functions
 - * Add functions to compare arrays of per-Plane values
 - RealmRttBaseEqual()
 - RealmVmidEqual()
 - * Modify signatures to accept array of per-Plane values
 - VmidIsFree() -> VmidsAreFree()
 - VmidIsValid() -> VmidsAreValid()
 - * Modify signature of RttWalk() to take an RTT tree index
- RMI_REALM_CREATE
 - * Add success conditions
 - rtt_tree_pp
 - num_aux_planes
- Add RMI_RTT_AUX_* commands for manipulation of auxiliary RTTs
- RMI_{DATA_CREATE,RTT_SET_RIPAS}: add failure condition if IPA is live in an auxiliary RTT
- Add RSI_PLANE_ENTER and describe Plane exception model
- Add RSI_MEM_* commands and describe memory access permission management
- RsiRealmConfig: add num_aux_planes
- RsiFeatureRegister0: add MRO field

Clarifications

- Realm device assignment
 - Replace references to DOE, ECAM, IDE with reference to PCIe 6.0
 - Amend “Realm device assignment overview” to correctly describe flow for retrieval of device interface report and device measurements
 - Amend “Device requests and responses” to include RSI_RDEV_GET_{INTERFACE_REPORT,MEASUREMENTS}
 - Amend IDE setup flow to include RSI_RDEV_{LOCK,GET_INTERFACE_REPORT,GET_MEASUREMENTS}
 - Clarify that RSI_RDEV_CONTINUE causes a REC exit due to IO, which must be completed by Host execution of RMI_VDEV_COMMUNICATE
 - Amend “Virtual device teardown flow” to show disabling of SMMU on RSI_VDEV_STOP
 - Remove incorrect statement that IPA base is present in the interface report
 - Amend description of RSI_RDEV_STOP: in-flight transactions may still complete, but future device accesses to Realm memory are blocked
 - RMI_PDEV_NOTIFY: add check for validity of event identifier
 - State that VDEV and RDEV are Host and Realm views of the same underlying RMM object
 - Describe relationship between RDEV state and TDI state
- Compress rendering of array struct members
- Consistently postfix name of “before: true” context values with “_pre”
- Replace inline “Realm(rd)” expressions with “realm” context value
- Replace inline “Rec(rec_ptr)” expressions with “rec” context value
- Improve consistency of function names
 - Casting a memory location to an object is suffixed “At”, for example “RecAt”
 - Casting an object to an interface type is suffixed “To”, for example “RttEntryStateToRmi”
 - Rename RttEntryFromDescriptor() to RttDescriptorDecode()
- RMI_RTT_READ_ENTRY: add ripas_prot success condition
- Feature discovery and selection

- Rename RmiFeatureRegister0 fields
 - * SVE_EN -> SVE
 - * PDEV_AUX -> PDEV_NUM_AUX
 - * PMU_EN -> PMU
- {RMI,RSI}_FEATURES: add {Rmi,Rsi}FeatureRegisterEncode() functions
 - * Formalise the mapping from features supported by the platform to the output value of these commands
- RMI_REALM_CREATE: implement RealmParamsSupported()
 - * Formalise the check that input values to this command are compatible with features supported by the platform

Defects

- Realm device assignment
 - RTT folding: for ASSIGNED_IO_*, require that memory attributes (which are Host-controlled) are the same for all entries
 - Permit Host to specify any Device memory attributes for an IO mapping
 - Specify that IO mappings are Outer Shareable
 - Remove RmiIoEnter::req_size, and state that the Host should always provide a 4KB request buffer
 - Add RmiIoExit::req_type, to inform Host via which mailbox to route the device request
 - RSI_RDEV_GET_DIGESTS: do not trigger REC exit due to IO
- Correct typo in “REC entry” section [ILFYDV]

Relaxations

- Realm device assignment
 - RMI_{IO_}_GRANULE_UNDELEGATE: permit new GPT entry to be any value except GPT_REALM
- RMI_RTT_{INIT,SET}_RIPAS: relax “top_rtt_align” failure condition
 - The previous condition caused the command to fail if the “top” address was misaligned
 - This is replaced with “no_progress”, which only fails if the command does not modify any RTT entries

1.1-alp0 (05-10-2023)

New features

- Realm device assignment (FENIMORE-722)
 - New Granule states: PDEV{AUX}, VDEV, IO{UNDELEGATED,DELEGATED_PRIVATE,DELEGATED_SHARED}, IO_{PRIVATE,SHARED}
 - * RMI commands for (un)delegation of IO physical memory
 - New RMM objects: PDEV, VDEV
 - * RMI commands for creation, destruction and lifecycle management
 - Realm-facing abstraction for an assigned device: RDEV
 - RMM-device communication flow (protected by SPDM)
 - * Mediated through “REC exit due to IO”, RMI_{PDEV,VDEV}_COMMUNICATE commands and buffers in NS memory
 - * When triggered by a Realm action, managed via an “interruptible operation” programming interface, similar to the existing one for RSI_ATTESTATION_TOKEN commands
 - Host caching of device attestation evidence (certificate, measurements, interface report), with integrity ensured via RMM-held digests
 - New HIPAS values (ASSIGNED_IO_PRIVATE, ASSIGNED_IO_SHARED) and RIPAS value (IO)
 - * Extension of rules regarding Realm access to Protected IPA space
 - * Extension of rules regarding RTT (un)folding
 - Extension of RIPAS change flow for the purpose of validating mappings against the device interface report, and transition to RIPAS IO

Clarifications

None

Defects

None

Relaxations

None

1.0-eac5 (05-10-2023)

Clarifications

- Fix attestation token flows (FENIMORE-718)
- Clarify behavior on Host rejection of a RIPAS change request (FENIMORE-719)
- Replace Granule::pas attribute with Granule::gpt
 - PAS is an attribute of a memory access, not of a Granule.

Defects

- {RMI,RSI}_VERSION: (FENIMORE-724)
 - Clarify rules regarding returned interface version, and provide examples
 - Remove rule that if the return code is SUCCESS, subsequent calls to the interface adhere to the behavior corresponding with the returned interface version
- Specify that SMCCC registers not specified as command input / output values are SBZ and MBZ respectively (FENIMORE-724)
- RSI_ATTESTATION_TOKEN_INIT: return upper bound on token size (FENIMORE-720)
- RMI_DATA_CREATE: move RIPAS=RAM from being a pre-condition to a post-condition (FENIMORE-721)

Relaxations

None

1.0-eac4 (06-09-2023)

Clarifications

- Exclude GIC, timer and PMU values from “On REC exit ... all other REC exit fields are zero” (FENIMORE-712)
- Amend contradictory statement regarding RTT folding to level 1 (FENIMORE-715) [IqwQSB]

Defects

- RMI_RTT_{INIT,SET}_RIPAS: fix “top” alignment check
 - Ensure that “top” is Granule aligned (FENIMORE-710)
 - Ensure that return code is deterministically specified (FENIMORE-711)
 - Prevent RIPAS change from proceeding beyond the “top” address provided by the Realm (FENIMORE-711)
- {RMI,RSI}_VERSION: add handshake (FENIMORE-708)
 - The caller provides a “requested version”
 - The RMM either returns:
 - * A version which it can implement, that is compatible with the requested version (and a SUCCESS return code)
 - * A version which it implements, that is incompatible with the requested version (and an error code)
 - If the return code is SUCCESS, subsequent calls to the interface adhere to the behavior corresponding with the returned interface version
- Increase width of PsciReturnCode to 64 bits (FENIMORE-709)

Relaxations

- RMI_REALM_CREATE: permit number of PMU counters to be less than number supported by the implementation (FENIMORE-716)
- RMI_REALM_CREATE: permit number of breakpoints or watchpoints to be less than number supported by the implementation (FENIMORE-717)

1.0-eac3 (20-07-2023)

Clarifications

- Clarify which bits of command input / output values should / must be zero (FENIMORE-674)
- Explain distinction between concrete and abstract types (FENIMORE-693)

- Clarify return value from RSI_IPA_STATE_SET when stopping at first DESTROYED entry (FENIMORE-699) [IGXDDX]

Defects

- PSCI_SYSTEM_{OFF,RESET}: change Realm state to SYSTEM_OFF (FENIMORE-694)
- RMI_REC_CREATE: update RIM only if runnable flag is set (FENIMORE-697)
- RMI_REALM_CREATE: fix list of measured parameters (FENIMORE-695)
- Remove members from RmmSystemRegisters (FENIMORE-700)
 - State saved / restored depends on architecture features supported by the platform, so defining this type as an empty placeholder
- Avoid use of reserved ASL v1 keyword “entry” in MRS (FENIMORE-702)
 - RmiRecEntry -> RmiRecEnter
 - RmiRecEntryFlags -> RmiRecEnterFlags
 - RmiRecRun::entry -> RmiRecRun::enter
 - RmmRttWalkResult::entry -> RmmRttWalkResult::rtte
- RSI_IPA_STATE_SET: prohibit RSI_DESTROYED input value (FENIMORE-705)
- RMI_PSCI_COMPLETE: PSCI_CPU_ON: fix copy of context_id to target CPU X0 (FENIMORE-703)
- Allow Host to reject request to change RIPAS to RAM (FENIMORE-661)
- Allow Host to reject PSCI_CPU_ON request via RMI_PSCI_COMPLETE (FENIMORE-706)

Relaxations

- Permit folding of level 2 RTT to create level 1 block mapping (FENIMORE-608)
- Remove restriction that attestation token size must not exceed 4KB (FENIMORE-691)

1.0-eac2 (07-06-2023)

Clarifications

- Remove reference to triggering ERROR_INPUT by setting MBZ bit to 1 (FENIMORE-675)
- Clarify constraints on output values in case of command failure [R_TFZMS] (FENIMORE-676)
- Clarify encoding of RmiRealmParams::sve_sz (FENIMORE-684)
- Clarify set of SMCCC interfaces available to a Realm [R_NPLKX] (FENIMORE-685)

Defects

- Replace PMU fields in RmiRecExit with single bit indicating the PMU overflow status [R_WXTZF] (FENIMORE-679)
- RMI_PSCI_COMPLETE: failure condition should compare against MPIDR, not RD address (FENIMORE-681)
- RMI_REC_CREATE: remove params_valid failure condition (FENIMORE-686)
- RMI_RTT_{INIT,SET}_RIPAS: check alignment of “top” input value (FENIMORE-687)
- Reduce coupling between HIPAS and RIPAS (FENIMORE-680)
 - Replace HIPAS=DESTROYED with RIPAS=DESTROYED
 - Remove RmiRttEntryState::RMI_DESTROYED
 - Change encoding of RmiRttEntryState::RMI_TABLE
 - Add RmiRipas::RMI_DESTROYED
 - Add RsiRipas::RSI_DESTROYED
 - RMI_DATA_CREATE_UNKNOWN: remove pre-condition that RIPAS=RAM
 - RMI_DATA_DESTROY:
 - * In all cases, post-condition now states that HIPAS=UNASSIGNED
 - * If pre-condition was RIPAS=RAM, post-condition states that RIPAS=DESTROYED
 - RMI_RTT_DESTROY:
 - * Remove post-condition that HIPAS=DESTROYED
 - * Add post-condition that state of parent RTTE is UNASSIGNED
 - * Add post-condition that RIPAS=DESTROYED
 - RMI_RTT_SET_IPA_STATE: stop at first DESTROYED entry if “destroyed” flag is set
 - RSI_IPA_STATE_SET: add “destroyed” flag
 - Clarify distinction between “RTT folding” [D_QPXCp] and “RTT destruction” [D_VXRZW]
- RMI_RTT_INIT_RIPAS: success conditions should be bounded by walk_top, not top

Relaxations

- RSI_REALM_CONFIG: provide Realm hash algorithm (FENIMORE-678)

1.0-eac1 (31-03-2023)

Clarifications

- Unused bits of RmiRecEntry::gicv3_hcr are SBZ [I_{SMHXB}] (FENIMORE-666)
- RMI_REC_ENTER: all RMI_ERROR_INPUT failure conditions precede all RMI_ERROR_REC failure conditions (FENIMORE-668)
- Avoid use of raw Xn values in command conditions where possible (FENIMORE-671)
- Clarify definition of REC exit due to (Non-)emulatable Data Abort [D_{CYRMT}, D_{MTZMC}] (FENIMORE-673)

Defects

- RMI_RTT_INIT_RIPAS: take account of “top” IPA value when calculating RIM contribution (FENIMORE-662)
- RttSkipEntriesWithRipas: fix inverted logic (FENIMORE-663)
- RMI_RTT_SET_RIPAS: on success, modify IPA range [base, walk_top) (FENIMORE-669)
- RMI_RTT_{INIT,SET}_RIPAS: remove redundant failure conditions (FENIMORE-670)
- Clarify HIPAS=DESTROYED implies RIPAS=UNDEFINED [R_{JYDRL}] (FENIMORE-672)

Relaxations

- RSI_HOST_CALL: relax alignment requirement from 4KB to 256B

1.0-eac0 (31-01-2023)

Clarifications

None

Defects

- RmiRealmParams: reduce width of integer attributes (FENIMORE-647)
- RSI_IPA_STATE_SET: replace (base, size) with (base, top) (FENIMORE-656)
- RMI_RTT_INIT_RIPAS, RMI_RTT_SET_RIPAS: allow single command to modify multiple RTT entries (FENIMORE-656)

Relaxations

- RMI_RTT_SET_RIPAS: remove “ripas” input value (FENIMORE-659)

1.0-bet2 (16-12-2022)

Clarifications

- Flows: update RMI_REC_ENTRY to take a single ‘run’ input value
- Clarify meaning of “TTD” [I_{YMNSR}] (FENIMORE-641)
- Fix typo in reference to “CCA platform token claim map” [I_{FJKFY}] (FENIMORE-647)
- Fix reference to “RME system architecture spec” (FENIMORE-648)
- Flows: remove stale reference to parameters passed to RMI_DATA_CREATE (FENIMORE-649)
- Improve definition and consistency of usage of the term “REC” (FENIMORE-650)
 - Where referring to the RMM data structure “REC object” is now used
- Clarify description of properties of Realm IPA space [I_{TPGKW}] (FENIMORE-639)
 - Replace “permitted, under control of host” with statements which refer to particular HIPAS values.
 - Add “Protected IPA, HIPAS=DESTROYED” row, thereby removing contradictory statements regarding SEA taken to Realm, previously in “Protected IPA, RIPAS=EMPTY”.
- On assertion of an EL1 timer, the RMM guarantees a *REC exit*, not only a *Realm exit* (FENIMORE-651)
- RMI_RTT_FOLD: preserve RIPAS value if IPA is Protected (FENIMORE-638)

Defects

- Attestation: wrap sub-tokens in byte stream (FENIMORE-643)
- RMI_DATA_DESTROY, RMI_RTT_{DESTROY,FOLD}: return PA of destroyed object (FENIMORE-563)

- RMI_REALM_DESTROY, RMI_REC_DESTROY, RMI_REC_ENTER, RMI_RTT_DESTROY, RMI_RTT_FOLD, RMI_RTT_SET_RIPAS: Remove RMI_ERROR_IN_USE (FENIMORE-588)
- RMI_DATA_CREATE, RMI_DATA_CREATE_UNKNOWN, RMI_REC_CREATE, RMI_RTT_CREATE: pass RD pointer in X1 (FENIMORE-655)
- Replace RmiRealmParams::features_0 with discrete fields (FENIMORE-655)
- RMI_DATA_CREATE(_UNKNOWN): require RIPAS=RAM (FENIMORE-645)
- Apply “must / should be zero” consistently (FENIMORE-619)
 - In command inputs, unused bits are SBZ
 - In command outputs, unused bits are MBZ

Relaxations

- RSI_HOST_CALL: expand set of GPRs to X0-X30 (FENIMORE-607)
 - This enables the RMM to support any calling convention.
- RMI_DATA_DESTROY, RMI_RTT_DESTROY, RMI_RTT_UNMAP_UNPROTECTED: return IPA of next live RTT entry (FENIMORE-563)

1.0-beta1 (31-10-2022)

Clarifications

- Rename HIPAS_VALID_NS -> UNASSIGNED (FENIMORE-631)
- SEA injection is independent of whether Host emulates MMIO (FENIMORE-632)
- In RIPAS change flow, permit Host to apply the change to zero or more pages of the target IPA region (FENIMORE-633)
- Flows: replace HVC with Host call (FENIMORE-611)
- Clarify behavior of VmidIsValid() function (FENIMORE-630)
- Qualify “all other exit fields are zero” statements [R_GTJRP, R_LRCP] (FENIMORE-634)
 - GIC, timer and PMU fields are valid on every REC exit.

Defects

- Change size of RsiHostCall type to 256 bytes (FENIMORE-629)
- Correct the set of ESR_EL2 fields which are returned to the Host on REC exit due to Data abort [R_RYVFL]
 - On all Data Aborts, add FnV.
 - On Emulatable Data Aborts, add SF.
 - On Non-emulatable Data Abort at an Unprotected IPA, add IL.

Relaxations

None

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 Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. No license, express, implied or otherwise, is granted to Licensee under this License, to use the Arm trade marks in connection with the Document or any products based thereon. Visit Arm's website at <http://www.arm.com/company/policies/trademarks> for more information about Arm's trademarks.

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

Copyright © 2022-2024 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

DRAFT

Contents

Realm Management Monitor specification

Realm Management Monitor specification	ii
Release information	iii
Arm Non-Confidential Document License ("License")	xvii

Preface

Quality level	xxxii
Conventions	xxxiii
Typographical conventions	xxxiii
Numbers	xxxiii
Pseudocode descriptions	xxxiii
Addresses	xxxiii
Rules-based writing	xxxiv
Content item identifiers	xxxiv
Content item rendering	xxxiv
Content item classes	xxxiv
Additional reading	xxxv
Feedback	xxxvii
Feedback on this book	xxxvii
Open issues	xxxviii

Part A Architecture

Chapter A1

Overview

A1.1 Confidential computing	40
A1.2 System software components	41
A1.3 Realm Management Monitor	41

Chapter A2

Concepts

A2.1 Realm	44
A2.1.1 Overview	44
A2.1.2 Realm execution environment	44
A2.1.3 Realm attributes	45
A2.1.4 Realm liveness	47
A2.1.5 Realm lifecycle	47
A2.1.6 Realm parameters	48
A2.1.7 Realm Descriptor	48
A2.2 Granule	50
A2.2.1 Granule category	50
A2.2.2 Granule attributes	51
A2.2.3 Granule lifecycle	51
A2.2.4 Granule ownership	55
A2.2.5 Granule wiping	55
A2.3 Realm Execution Context	57
A2.3.1 Overview	57
A2.3.2 REC attributes	57
A2.3.3 REC index and MPIDR value	59
A2.3.4 REC lifecycle	59

Chapter A3**Feature discovery and configuration**

A3.1	Feature discovery and configuration overview	62
A3.2	Realm hash algorithm	62
A3.3	Realm LPA2 and IPA width	62
A3.4	Realm support for Scalable Vector Extension	63
A3.5	Realm support for self-hosted debug	64
A3.6	Realm support for Performance Monitors Extension	64
A3.7	Realm support for Activity Monitors Extension	64
A3.8	Realm support for Statistical Profiling Extension	64
A3.9	Realm support for Trace Buffer Extension	64
A3.10	Support for Realm device assignment	65
A3.11	Support for auxiliary Planes	65
A3.12	Support for Stage 2 Access Permissions indirect encoding	65
A3.13	Live Firmware Activation	66
A3.14	GICv3 virtualization	66
A3.15	Support for Realm memory encryption	66

Chapter A4**Realm exception model**

A4.1	Realm exception model overview	68
A4.2	REC entry	70
	A4.2.1 RmiRecEnter object	70
	A4.2.2 General purpose registers restored on REC entry	70
	A4.2.3 REC entry following REC exit due to Data Abort	71
A4.3	REC exit	72
	A4.3.1 RmiRecExit object	72
	A4.3.2 Realm exit reason	73
	A4.3.3 General purpose registers saved on REC exit	73
	A4.3.4 REC exit due to synchronous exception	74
	A4.3.5 REC exit due to IRQ	77
	A4.3.6 REC exit due to FIQ	77
	A4.3.7 REC exit due to PSCI	77
	A4.3.8 REC exit due to RIPAS change pending	78
	A4.3.9 REC exit due to Host call	78
	A4.3.10 REC exit due to SError	79
	A4.3.11 REC exit due to S2AP change pending	79
	A4.3.12 REC exit due to VDEV request	80
	A4.3.13 REC exit due to VDEV communication	80
	A4.3.14 REC exit due to device memory mapping validation	80
A4.4	Emulated Data Aborts	82
A4.5	Host call	82

Chapter A5**Realm memory management**

A5.1	Realm memory management overview	84
A5.2	Realm view of memory management	84
	A5.2.1 Realm IPA space	84
	A5.2.2 Realm IPA state	84
	A5.2.3 Realm access to a Protected IPA	85
	A5.2.4 RSI command access to a Protected IPA	85
	A5.2.5 Changes to RIPAS while Realm state is REALM_NEW	85
	A5.2.6 Changes to RIPAS while Realm state is REALM_ACTIVE	86
	A5.2.7 Realm access to an Unprotected IPA	87
	A5.2.8 Synchronous External Aborts	88
	A5.2.9 Realm access outside IPA space	88
	A5.2.10 Summary of Realm IPA space properties	89
	A5.2.11 Cache maintenance operations	89

A5.3	Host view of memory management	90
A5.3.1	Host IPA state	90
A5.3.2	Changes to HIPAS while Realm state is REALM_NEW	92
A5.3.3	Changes to HIPAS while Realm state is REALM_ACTIVE	92
A5.3.4	Summary of changes to HIPAS and RIPAS of a Protected IPA	93
A5.3.5	Dependency of RMI command execution on RIPAS and HIPAS values	96
A5.3.6	Changes to HIPAS of an Unprotected IPA	97
A5.4	RIPAS change	98
A5.5	Device memory mapping validation	100
A5.6	Realm Translation Table	102
A5.6.1	RTT overview	102
A5.6.2	RTT structure and configuration	102
A5.6.3	RTT starting level	102
A5.6.4	RTT entry	103
A5.6.5	RTT reading	104
A5.6.6	RTT folding	104
A5.6.7	RTT unfolding	105
A5.6.8	RTTE liveness and RTT liveness	105
A5.6.9	RTT destruction	106
A5.6.10	RTT walk	106
A5.6.11	Stage 2 Access Permissions	107
A5.6.12	Memory attributes	109
Chapter A6	Realm interrupts and timers	
A6.1	Realm interrupts	113
A6.2	Realm timers	116
Chapter A7	Realm measurement and attestation	
A7.1	Realm measurements	119
A7.1.1	Realm Initial Measurement	119
A7.1.2	Realm Extensible Measurement	120
A7.2	Realm attestation	121
A7.2.1	Attestation token	121
A7.2.2	Attestation token generation	121
A7.2.3	Attestation token format	123
Chapter A8	Realm debug and performance monitoring	
A8.1	Realm PMU	144
Chapter A9	Realm device assignment	
A9.1	Realm device assignment overview	146
A9.1.1	Device protection types	146
A9.1.2	Device attestation types	146
A9.2	Communication between RMM and a device	151
A9.2.1	Device requests and responses	151
A9.2.2	Mapping from virtual device ID to VDEV object	152
A9.2.3	Host-side device communication flow	154
A9.2.4	Host caching of device attestation evidence	159
A9.2.5	Device communication data structures	159
A9.3	Physical device object	161
A9.3.1	Physical device attributes	161
A9.3.2	Physical device lifecycle	162
A9.3.3	Physical device flows	166
A9.4	Virtual device object	172
A9.4.1	Virtual device attributes	172

A9.4.2	Virtual device invariants	172
A9.4.3	Virtual device lifecycle	173
A9.4.4	Virtual device flows	176
A9.5	Realm management of an assigned device interface	177
A9.5.1	Interruptible Realm device operations	177
A9.5.2	Realm retrieval of device attestation evidence	178
A9.5.3	Realm validation of device memory mappings	178
A9.5.4	Realm device attributes	180
A9.5.5	Realm device lifecycle	181
A9.5.6	Realm device flows	186
A9.6	Virtual SMMU	187
A9.6.1	VSMMU lifecycle	187
A9.6.2	VSMMU attributes	187
A9.6.3	VSMMU liveness	188
A9.6.4	VSMMU validation	188
A9.6.5	PSMMU interrupts	189
A9.7	Device access to a Protected IPA	190

Chapter A10

Planes

A10.1	Planes overview	192
A10.2	Planes exception model	193
A10.2.1	Plane exception model overview	193
A10.2.2	Plane entry	193
A10.2.3	Plane exit	194
A10.2.4	REC exit from Pn	195
A10.2.5	Pn execution of HVC and SMC	196
A10.2.6	Pn system registers	196
A10.3	Planes memory management	197
A10.3.1	Auxiliary RTT	197
A10.3.2	Stage 2 Access Permissions within a multi-Plane Realm	198
A10.3.3	Stage 2 Access Permissions within a multi-Plane Realm overview	198
A10.4	Planes interrupts	202
A10.5	Planes timers	203

Chapter A11

Realm memory encryption

Part B Interface

Chapter B1

Commands

B1.1	Overview	210
B1.2	Command definition	211
B1.2.1	Example command	211
B1.3	Command registers	212
B1.4	Command condition expressions	212
B1.5	Command context values	213
B1.6	Command failure conditions	214
B1.7	Command success conditions	215
B1.8	Concrete and abstract types	215
B1.9	Command footprint	215
B1.10	Command testing	216

Chapter B2

Interface versioning

Chapter B3

Command condition functions

B3.1	AddrInRange function	221
------	----------------------	-----

B3.2	AddrIsAligned function	221
B3.3	AddrIsAuxLive function	222
B3.4	AddrIsGranuleAligned function	222
B3.5	AddrIsProtected function	222
B3.6	AddrIsRttLevelAligned function	222
B3.7	AddrIsWithin function	222
B3.8	AddrRangelsAuxLive function	223
B3.9	AddrRangelsProtected function	223
B3.10	AddrRangelsWithin function	223
B3.11	AlignDownToRttLevel function	223
B3.12	AlignUpToRttLevel function	223
B3.13	AuxAlias16 function	224
B3.14	AuxAlias32 function	224
B3.15	AuxAligned16 function	224
B3.16	AuxAligned32 function	225
B3.17	AuxEqual16 function	225
B3.18	AuxEqual32 function	225
B3.19	AuxSort function	226
B3.20	AuxStateEqual16 function	226
B3.21	AuxStateEqual32 function	226
B3.22	AuxStates function	226
B3.23	CurrentRealm function	227
B3.24	CurrentRec function	227
B3.25	DeviceCommunicate function	227
B3.26	Equal function	227
B3.27	FeatureToRmi function	229
B3.28	FeatureToRsi function	229
B3.29	Gicv3ConfigIsValid function	230
B3.30	GranuleAccessPermitted function	230
B3.31	GranuleAt function	230
B3.32	ImplFeatures function	230
B3.33	MecMembers function	231
B3.34	MecPolicy function	231
B3.35	MecState function	231
B3.36	MemPermLabelSupported function	231
B3.37	MinAddress function	231
B3.38	MpidrEqual function	232
B3.39	MpidrUsed function	232
B3.40	MsiAddrIsValid function	232
B3.41	PalsDelegable function	232
B3.42	PalsDelegableCohDevMem function	232
B3.43	PalsDelegableDevMem function	232
B3.44	PalsDelegableDram function	233
B3.45	PalsDelegableNonCohDevMem function	233
B3.46	PdevAt function	233
B3.47	PdevAuxCount function	233
B3.48	PdevFlags function	233
B3.49	PdevVsmmulCompatible function	233
B3.50	PlaneRegIsValid function	234
B3.51	PlaneRegValue function	234
B3.52	PsciReturnCodeEncode function	234
B3.53	PsciReturnCodePermitted function	234
B3.54	PsmmuAddrIsValid function	235
B3.55	PsmmuSupportsMsi function	235
B3.56	RdevFromInstId function	235

B3.57	RdevFromVdevId function	235
B3.58	RdevIdsValid function	235
B3.59	RdevIdsAreValid function	235
B3.60	ReadMemory function	235
B3.61	RealmAt function	236
B3.62	RealmsLive function	236
B3.63	RealmParamsSupported function	236
B3.64	RealmRttBaseEqual function	238
B3.65	RealmVmidEqual function	238
B3.66	RecAt function	238
B3.67	RecAuxCount function	238
B3.68	RecDevMemResponseToRsi function	239
B3.69	RecFromMpidr function	239
B3.70	RecIndex function	239
B3.71	RecRipasResponseToRsi function	239
B3.72	RecS2APResponseToRsi function	240
B3.73	RemExtend function	240
B3.74	ResultEqual function	240
B3.75	RimExtendData function	241
B3.76	RimExtendRec function	241
B3.77	RimExtendRipas function	241
B3.78	RimExtendRipasForEntry function	241
B3.79	RimInit function	242
B3.80	RipasToRmi function	242
B3.81	RmiAddressRangesEqual16 function	242
B3.82	RmiAddressRangesEqual4 function	242
B3.83	RmiDevCommDataAt function	243
B3.84	RmiFeatureRegister0Decode function	243
B3.85	RmiFeatureRegisterEncode function	243
B3.86	RmiPdevEventsValid function	244
B3.87	RmiPdevFlagsDecode function	244
B3.88	RmiPdevParamsAt function	244
B3.89	RmiPdevParamsValid function	245
B3.90	RmiPublicKeyParamsAt function	245
B3.91	RmiRealmParamsAt function	245
B3.92	RmiRealmParamsValid function	245
B3.93	RmiRecParamsAt function	245
B3.94	RmiRecRunAt function	246
B3.95	RmiVdevFlagsDecode function	246
B3.96	RmiVdevParamsAt function	246
B3.97	RmiVdevParamsValid function	246
B3.98	RmiVersionHigherIsSupported function	246
B3.99	RmiVersionHighest function	246
B3.100	RmiVersionHighestBelow function	247
B3.101	RmiVersionIsSupported function	247
B3.102	RmiVersionLowerIsSupported function	247
B3.103	RmiVsmmuParamsAt function	247
B3.104	RmiVsmmuParamsValid function	247
B3.105	RsiDeviceInfoAt function	248
B3.106	RsiFeatureRegisterEncode function	248
B3.107	RsiHostCallAt function	248
B3.108	RsiPlaneRunAt function	248
B3.109	RsiRealmConfigAt function	248
B3.110	RsiVersionHigherIsSupported function	249
B3.111	RsiVersionHighest function	249

B3.112	RsiVersionHighestBelow function	249
B3.113	RsiVersionIsSupported function	249
B3.114	RsiVersionLowerIsSupported function	249
B3.115	RttAllEntriesContiguous function	250
B3.116	RttAllEntriesRipas function	250
B3.117	RttAllEntriesState function	250
B3.118	RttAt function	250
B3.119	RttConfigIsValid function	250
B3.120	RttDescriptorDecode function	251
B3.121	RttDescriptorIsValidForUnprotected function	251
B3.122	RttEntriesInRangeCohDevMem function	251
B3.123	RttEntriesInRangeMemAttr function	251
B3.124	RttEntriesInRangeNonCohDevMem function	252
B3.125	RttEntriesInRangeOutputContiguous function	252
B3.126	RttEntriesInRangeRipas function	253
B3.127	RttEntryAt function	253
B3.128	RttEntryIndex function	254
B3.129	RttEntryStateToRmi function	254
B3.130	RttFold function	254
B3.131	RttIsHomogeneous function	254
B3.132	RttIsLive function	254
B3.133	RttLevellsBlockOrPage function	255
B3.134	RttLevellsStarting function	255
B3.135	RttLevellsValid function	255
B3.136	RttLevelSize function	255
B3.137	RttMemAttrEqual function	256
B3.138	RttS2APEqual function	256
B3.139	RttsAllProtectedEntriesRipas function	256
B3.140	RttsAllProtectedEntriesState function	256
B3.141	RttsAllUnprotectedEntriesState function	257
B3.142	RttsGranuleState function	257
B3.143	RttSkipEntriesIfNotState function	257
B3.144	RttSkipEntriesUnlessRipas function	257
B3.145	RttSkipEntriesUnlessState function	257
B3.146	RttSkipEntriesWithRipas function	258
B3.147	RttSkipNonLiveEntries function	258
B3.148	RttsStateEqual function	259
B3.149	RttWalk function	259
B3.150	RttWalkAnyNotAligned function	259
B3.151	TdIdsFree function	260
B3.152	ToAddress function	260
B3.153	ToBits64 function	260
B3.154	VdevAt function	260
B3.155	VdevAuxCount function	260
B3.156	VersionEqual function	260
B3.157	VmidsAreFree function	261
B3.158	VmidsAreValid function	261
B3.159	VsidsFree function	261
B3.160	VsmmuAt function	261
B3.161	VsmmulsLive function	262

Chapter B4

Realm Management Interface

B4.1	RMI version	264
B4.2	RMI command return codes	264
B4.3	RMI commands	265

B4.3.1	RMI_DATA_CREATE command	267
B4.3.2	RMI_DATA_CREATE_UNKNOWN command	270
B4.3.3	RMI_DATA_DESTROY command	273
B4.3.4	RMI_DEV_MEM_MAP command	276
B4.3.5	RMI_DEV_MEM_UNMAP command	279
B4.3.6	RMI_FEATURES command	282
B4.3.7	RMI_GRANULE_DELEGATE command	283
B4.3.8	RMI_GRANULE_UNDELEGATE command	285
B4.3.9	RMI_MEC_SET_PRIVATE command	287
B4.3.10	RMI_MEC_SET_SHARED command	288
B4.3.11	RMI_PDEV_ABORT command	289
B4.3.12	RMI_PDEV_AUX_COUNT command	291
B4.3.13	RMI_PDEV_COMMUNICATE command	292
B4.3.14	RMI_PDEV_CREATE command	295
B4.3.15	RMI_PDEV_DESTROY command	298
B4.3.16	RMI_PDEV_GET_STATE command	300
B4.3.17	RMI_PDEV_IDE_RESET command	302
B4.3.18	RMI_PDEV_NOTIFY command	304
B4.3.19	RMI_PDEV_SET_PUBKEY command	306
B4.3.20	RMI_PDEV_STOP command	308
B4.3.21	RMI_PSCI_COMPLETE command	310
B4.3.22	RMI_PSMU_IRQ_NOTIFY command	314
B4.3.23	RMI_PSMU_MSI_CONFIG command	316
B4.3.24	RMI_REALM_ACTIVATE command	318
B4.3.25	RMI_REALM_CREATE command	320
B4.3.26	RMI_REALM_DESTROY command	324
B4.3.27	RMI_REC_AUX_COUNT command	326
B4.3.28	RMI_REC_CREATE command	327
B4.3.29	RMI_REC_DESTROY command	331
B4.3.30	RMI_REC_ENTER command	333
B4.3.31	RMI_RTT_AUX_CREATE command	336
B4.3.32	RMI_RTT_AUX_DESTROY command	339
B4.3.33	RMI_RTT_AUX_FOLD command	342
B4.3.34	RMI_RTT_AUX_MAP_PROTECTED command	345
B4.3.35	RMI_RTT_AUX_MAP_UNPROTECTED command	349
B4.3.36	RMI_RTT_AUX_UNMAP_PROTECTED command	352
B4.3.37	RMI_RTT_AUX_UNMAP_UNPROTECTED command	355
B4.3.38	RMI_RTT_CREATE command	358
B4.3.39	RMI_RTT_DESTROY command	361
B4.3.40	RMI_RTT_DEV_MEM_VALIDATE command	364
B4.3.41	RMI_RTT_FOLD command	368
B4.3.42	RMI_RTT_INIT_RIPAS command	371
B4.3.43	RMI_RTT_MAP_UNPROTECTED command	374
B4.3.44	RMI_RTT_READ_ENTRY command	377
B4.3.45	RMI_RTT_SET_RIPAS command	380
B4.3.46	RMI_RTT_SET_S2AP command	383
B4.3.47	RMI_RTT_UNMAP_UNPROTECTED command	386
B4.3.48	RMI_VDEV_ABORT command	389
B4.3.49	RMI_VDEV_AUX_COUNT command	391
B4.3.50	RMI_VDEV_COMMUNICATE command	392
B4.3.51	RMI_VDEV_COMPLETE command	395
B4.3.52	RMI_VDEV_CREATE command	397
B4.3.53	RMI_VDEV_DESTROY command	401
B4.3.54	RMI_VDEV_GET_STATE command	404
B4.3.55	RMI_VDEV_STOP command	406

B4.3.56	RMI_VERSION command	408
B4.3.57	RMI_VSMMU_CREATE command	410
B4.3.58	RMI_VSMMU_DESTROY command	413
B4.3.59	RMI_VSMMU_MAP command	415
B4.3.60	RMI_VSMMU_UNMAP command	418
B4.4	RMI types	421
B4.4.1	RmiAddressRange type	421
B4.4.2	RmiBoolean type	421
B4.4.3	RmiCommandReturnCode type	421
B4.4.4	RmiDataFlags type	422
B4.4.5	RmiDataMeasureContent type	422
B4.4.6	RmiDevCommData type	423
B4.4.7	RmiDevCommEnter type	423
B4.4.8	RmiDevCommExit type	424
B4.4.9	RmiDevCommExitFlags type	424
B4.4.10	RmiDevCommProtocol type	425
B4.4.11	RmiDevCommStatus type	426
B4.4.12	RmiEmulatedMmio type	426
B4.4.13	RmiFeature type	427
B4.4.14	RmiFeatureRegister0 type	427
B4.4.15	RmiFeatureRegister1 type	428
B4.4.16	RmiHashAlgorithm type	429
B4.4.17	RmiInjectSea type	429
B4.4.18	RmiInterfaceVersion type	430
B4.4.19	RmiLfaPolicy type	430
B4.4.20	RmiPdevEvent type	431
B4.4.21	RmiPdevFlags type	431
B4.4.22	RmiPdevParams type	432
B4.4.23	RmiPdevProtection type	433
B4.4.24	RmiPdevState type	433
B4.4.25	RmiPmuOverflowStatus type	434
B4.4.26	RmiPublicKeyParams type	434
B4.4.27	RmiRealmFlags0 type	435
B4.4.28	RmiRealmFlags1 type	436
B4.4.29	RmiRealmParams type	436
B4.4.30	RmiRecCreateFlags type	437
B4.4.31	RmiRecEnter type	438
B4.4.32	RmiRecEnterFlags type	439
B4.4.33	RmiRecExit type	439
B4.4.34	RmiRecExitReason type	441
B4.4.35	RmiRecMpidr type	442
B4.4.36	RmiRecParams type	442
B4.4.37	RmiRecRun type	443
B4.4.38	RmiRecRunnable type	443
B4.4.39	RmiResponse type	444
B4.4.40	RmiRipas type	444
B4.4.41	RmiRttEntryState type	444
B4.4.42	RmiRttPlaneFeature type	445
B4.4.43	RmiRttS2APEncoding type	445
B4.4.44	RmiSignatureAlgorithm type	446
B4.4.45	RmiSmmuAction type	446
B4.4.46	RmiSmmulrq type	447
B4.4.47	RmiStatusCode type	447
B4.4.48	RmiTrap type	448
B4.4.49	RmiVdevAction type	448

B4.4.50	RmiVdevFlags type	448
B4.4.51	RmiVdevParams type	449
B4.4.52	RmiVdevState type	450
B4.4.53	RmiVsmmuFlags type	450
B4.4.54	RmiVsmmuParams type	450

Chapter B5

Realm Services Interface

B5.1	RSI version	453
B5.2	RSI command return codes	453
B5.3	RSI commands	454
B5.3.1	RSI_ATTESTATION_TOKEN_CONTINUE command	455
B5.3.2	RSI_ATTESTATION_TOKEN_INIT command	457
B5.3.3	RSI_FEATURES command	459
B5.3.4	RSI_HOST_CALL command	460
B5.3.5	RSI_IPA_STATE_GET command	462
B5.3.6	RSI_IPA_STATE_SET command	464
B5.3.7	RSI_MEASUREMENT_EXTEND command	466
B5.3.8	RSI_MEASUREMENT_READ command	468
B5.3.9	RSI_MEM_GET_PERM_VALUE command	470
B5.3.10	RSI_MEM_SET_PERM_INDEX command	472
B5.3.11	RSI_MEM_SET_PERM_VALUE command	474
B5.3.12	RSI_PLANE_ENTER command	476
B5.3.13	RSI_PLANE_REG_READ command	478
B5.3.14	RSI_PLANE_REG_WRITE command	480
B5.3.15	RSI_RDEV_CONTINUE command	482
B5.3.16	RSI_RDEV_GET_INFO command	485
B5.3.17	RSI_RDEV_GET_INSTANCE_ID command	487
B5.3.18	RSI_RDEV_GET_INTERFACE_REPORT command	489
B5.3.19	RSI_RDEV_GET_MEASUREMENTS command	492
B5.3.20	RSI_RDEV_GET_STATE command	495
B5.3.21	RSI_RDEV_LOCK command	497
B5.3.22	RSI_RDEV_START command	499
B5.3.23	RSI_RDEV_STOP command	501
B5.3.24	RSI_RDEV_VALIDATE_MAPPING command	503
B5.3.25	RSI_REALM_CONFIG command	505
B5.3.26	RSI_VERSION command	507
B5.3.27	RSI_VSMMU_ACTIVATE command	509
B5.3.28	RSI_VSMMU_GET_INFO command	511
B5.4	RSI types	513
B5.4.1	RsiBoolean type	513
B5.4.2	RsiCommandReturnCode type	513
B5.4.3	RsiDeviceAttestationType type	513
B5.4.4	RsiDeviceInfo type	514
B5.4.5	RsiDeviceState type	514
B5.4.6	RsiDevMemCoherent type	515
B5.4.7	RsiDevMemFlags type	515
B5.4.8	RsiDevMemOrdering type	516
B5.4.9	RsiFeature type	516
B5.4.10	RsiFeatureRegister0 type	517
B5.4.11	RsiGicOwner type	517
B5.4.12	RsiHashAlgorithm type	518
B5.4.13	RsiHostCall type	518
B5.4.14	RsiInterfaceVersion type	519
B5.4.15	RsiPlaneEnter type	519
B5.4.16	RsiPlaneEnterFlags type	520

B5.4.17	RsiPlaneExit type	520
B5.4.18	RsiPlaneExitReason type	521
B5.4.19	RsiPlaneRun type	522
B5.4.20	RsiRealmConfig type	522
B5.4.21	RsiResponse type	522
B5.4.22	RsiRipas type	523
B5.4.23	RsiRipasChangeDestroyed type	523
B5.4.24	RsiRipasChangeFlags type	523
B5.4.25	RsiTrap type	524

Chapter B6

Power State Control Interface

B6.1	PSCI overview	526
B6.2	PSCI version	526
B6.3	PSCI commands	527
B6.3.1	PSCI_AFFINITY_INFO command	528
B6.3.2	PSCI_CPU_OFF command	530
B6.3.3	PSCI_CPU_ON command	531
B6.3.4	PSCI_CPU_SUSPEND command	533
B6.3.5	PSCI_FEATURES command	534
B6.3.6	PSCI_SYSTEM_OFF command	535
B6.3.7	PSCI_SYSTEM_RESET command	536
B6.3.8	PSCI_VERSION command	537
B6.4	PSCI types	538
B6.4.1	PsciInterfaceVersion type	538
B6.4.2	PsciReturnCode type	538

Part C Constants and types

Chapter C1

RMM constants

C1.1	RMM_GRANULE_SIZE	541
C1.2	RMM_NUM_PERM_OVERLAY_INDICES	541
C1.3	RMM_RTT_BLOCK_LEVEL	541
C1.4	RMM_RTT_PAGE_LEVEL	541
C1.5	RMM_RTT_TREE_PRIMARY	542

Chapter C2

RMM types

C2.1	RmmAddressRange type	543
C2.2	RmmBoolean type	543
C2.3	RmmDataFlags type	544
C2.4	RmmDataMeasureContent type	544
C2.5	RmmDevCommState type	545
C2.6	RmmDevMemCoherent type	545
C2.7	RmmDevMemFlags type	546
C2.8	RmmDevMemOrdering type	546
C2.9	RmmFeature type	546
C2.10	RmmFeatures type	547
C2.11	RmmGptEntry type	548
C2.12	RmmGranule type	548
C2.13	RmmGranuleState type	548
C2.14	RmmHashAlgorithm type	549
C2.15	RmmHipas type	549
C2.16	RmmLfaPolicy type	550
C2.17	RmmMeasurementDescriptorData type	550
C2.18	RmmMeasurementDescriptorRec type	551

C2.19	RmmMeasurementDescriptorRipas type	551
C2.20	RmmMecPolicy type	552
C2.21	RmmMecState type	552
C2.22	RmmMemPermLocked type	552
C2.23	RmmMemPerms type	553
C2.24	RmmPdev type	553
C2.25	RmmPdevProtection type	554
C2.26	RmmPdevState type	554
C2.27	RmmPhysicalAddressSpace type	555
C2.28	RmmRdev type	555
C2.29	RmmRdevOperation type	556
C2.30	RmmRdevState type	556
C2.31	RmmRealm type	557
C2.32	RmmRealmMeasurement type	558
C2.33	RmmRealmState type	558
C2.34	RmmRec type	558
C2.35	RmmRecAttestState type	560
C2.36	RmmRecEmulatableAbort type	560
C2.37	RmmRecFlags type	560
C2.38	RmmRecPending type	561
C2.39	RmmRecResponse type	561
C2.40	RmmRecRunnable type	561
C2.41	RmmRecState type	562
C2.42	RmmRipas type	562
C2.43	RmmRipasChangeDestroyed type	562
C2.44	RmmRtt type	563
C2.45	RmmRttEntry type	563
C2.46	RmmRttEntryState type	564
C2.47	RmmRttMemAttr type	565
C2.48	RmmRttPlaneFeature type	565
C2.49	RmmRttProtected type	566
C2.50	RmmRttS2APDirect type	566
C2.51	RmmRttS2APEncoding type	566
C2.52	RmmRttS2APIndirect type	567
C2.53	RmmRttShareability type	567
C2.54	RmmRttWalkNotAligned type	567
C2.55	RmmRttWalkResult type	568
C2.56	RmmSystemRegisters type	568
C2.57	RmmVdev type	568
C2.58	RmmVdevState type	569
C2.59	RmmVsmmu type	569

Chapter C3

Generic types

C3.1	Address type	571
C3.2	BitsN type	571
C3.3	IntN type	571
C3.4	UIntN type	572

Part D Usage

Chapter D1

Flows

D1.1	Granule delegation flows	575
D1.1.1	Granule delegation flow	575
D1.1.2	Granule undelegation flow	575

D1.2	Realm lifecycle flows	577
D1.2.1	Realm creation flow	577
D1.2.2	Realm Translation Table creation flow	577
D1.2.3	Initialize memory of New Realm flow	578
D1.2.4	REC creation flow	580
D1.2.5	Realm destruction flow	582
D1.3	Realm exception model flows	584
D1.3.1	Realm entry and exit flow	584
D1.3.2	Host call flow	584
D1.3.3	REC exit due to Data Abort fault flow	585
D1.3.4	MMIO emulation flow	586
D1.4	PSCI flows	588
D1.4.1	PSCI_CPU_ON flow	588
D1.5	Realm memory management flows	591
D1.5.1	Add memory to Active Realm flow	591
D1.5.2	NS memory flow	591
D1.5.3	RIPAS change flow	592
D1.5.4	S2AP change flow	593
D1.6	Realm interrupts and timers flows	595
D1.6.1	Interrupt flow	595
D1.6.2	Timer interrupt delivery flow	595
D1.7	Realm attestation flows	597
D1.7.1	Attestation token generation flow	597
D1.7.2	Handling interrupts during attestation token generation flow	597
D1.8	Realm device assignment flows	599

Chapter D2

Realm shared memory protocol

D2.1	Realm shared memory protocol description	601
D2.2	Realm shared memory protocol flow	601

Glossary

Preface

Quality level

This table below summarises the quality level of the features which have been added in version 1.1 of this specification.

Feature	Quality level
Realm device assignment (foundation)	BETA
Realm device assignment (stage 1 SMMU)	ALPHA
Planes	BETA
Realm memory encryption	BETA
Live firmware activation	ALPHA
Platform software component countersigners	BETA

Due to the fact that some features are at ALPHA, the overall quality level of this version of the specification is ALPHA.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

`monospace`

Used for pseudocode and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in pseudocode and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example <http://developer.arm.com>

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by 0b, and hexadecimal numbers by 0x. In both cases, the prefix and the associated value are written in a monospace font, for example 0xFFFF0000. To improve readability, long numbers can be written with an underscore separator between every four characters, for example 0xFFFF_0000_0000_0000. Ignore any underscores when interpreting the value of a number.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Addresses

Unless otherwise stated, the term *address* in this specification refers to a physical address.

Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Declaration
- Rule
- Goal
- Information
- Rationale
- Implementation note
- Software usage

Declarations and Rules are normative statements. An implementation that is compliant with this specification must conform to all Declarations and Rules in this specification that apply to that implementation.

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and Rules, these are generally grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Content items other than Declarations and Rules are informative statements. These are provided as an aid to understanding this specification.

Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: L_{iiii}

- L is a label that indicates the content class of the content item.
- $iiii$ is the identifier of the content item.

Content item classes

Declaration

A Declaration is a statement that does one or more of the following:

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behaviour.

A Declaration is rendered with the label D .

Rule

A Rule is a statement that describes the behaviour of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label *R*.

Goal

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label *G*.

Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label *I*.

Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Developer (<http://developer.arm.com>) for access to Arm documentation.

- [1] *Introducing Arm CCA*. (ARM DEN 0125) Arm Limited.
- [2] *Arm Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A*. (ARM DDI 0615 A.d) Arm Ltd.
- [3] *Arm Architecture Reference Manual for A-Profile architecture*. (ARM DDI 0487 I.a) Arm Ltd.
- [4] *Arm CCA Security model*. (ARM DEN 0096) Arm Limited.
- [5] *Live Firmware Activation SMC Interface*. (ARM DEN 0147) Arm Limited.
- [6] *Arm Generic Interrupt Controller (GIC) Architecture Specification version 3 and version 4*. (ARM IHI 0069 G) Arm Ltd.
- [7] *Concise Binary Object Representation (CBOR)*. See <https://tools.ietf.org/html/rfc7049>
- [8] *CBOR Object Signing and Encryption (COSE)*. See <https://tools.ietf.org/html/rfc8152>
- [9] *Entity Attestation Token (EAT)*. See <https://datatracker.ietf.org/doc/draft-ietf-rats-eat/>

- [10] *Concise Data Definition Language (CDDL)*. See <https://tools.ietf.org/html/rfc8610>
- [11] *IANA Named Information Hash Algorithm Registry*. See <http://www.iana.org/assignments/named-information>
- [12] *SEC 1: Elliptic Curve Cryptography, version 2.0*. See <https://www.secg.org/sec1-v2.pdf>
- [13] *RME system architecture spec.* (ARM DEN 0129) Arm Ltd.
- [14] *PCI Express 6.0 specification*. See <https://pcisig.com/pci-express-6.0-specification>
- [15] *Secured Messages using SPDm Specification version 1.1.0*. See https://www.dmtf.org/sites/default/files/standards/documents/DSP0277_1.1.0.pdf
- [16] *Arm System Memory Management Unit Architecture Specification*. (ARM IHI 0070) Arm Limited.
- [17] *Arm SMC Calling Convention*. (ARM DEN 0028 D) Arm Ltd.
- [18] *Arm Specification Language Reference Manual*. (ARM DDI 0612 00bet7) Arm Ltd.
- [19] *Security Protocol and Data Model (SPDM)*. See <https://www.dmtf.org/dsp/DSP0274>
- [20] *Secure Hash Standard (SHS)*. See <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>
- [21] *RSA Cryptography Specifications Version 2.2*. See <https://datatracker.ietf.org/doc/rfc8017/>
- [22] *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*. See <https://datatracker.ietf.org/doc/html/rfc6979>
- [23] *Arm Power State Coordination Interface (PSCI)*. (ARM DEN 0022 D.b) Arm Ltd.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

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 (Realm Management Monitor specification).
- The number (DEN0137 1.1-alp11).
- The section name(s) to which your comments refer.
- The page number(s) to which your comments apply.
- The rule identifier(s) to which your comments apply, if applicable.
- 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.

Open issues

The following table lists known open issues in this version of the document.

Key	Description
-	Consider how teardown of DRAM mappings (via RMI_DATA_DESTROY) composes with teardown of device memory mappings (via RMI_DEV_MEM_UNMAP). In each case, the command returns the IPA of the next live entry - but it doesn't tell the caller whether this is DRAM or IO. How then can the caller know which of the two commands to call next, while still avoiding a (race-prone) call to RMI_RTT_READ_ENTRY?
-	In RMI_RTT_DEV_MEM_VALIDATE, consider how to combine: <ul style="list-style-type: none">• Modification of a range of RTT entries in a single command, and• Checking of output address and HIPAS values against rec.ripas_dev_pa.

DRAFT

DRAFT

Part A
Architecture

Chapter A1

Overview

The RMM is a software component which forms part of a system which implements the Arm Confidential Compute Architecture (Arm CCA). Arm CCA is an architecture which provides protected execution environments called *Realms*.

The threat model which Arm CCA is designed to address is described in [Introducing Arm CCA \[1\]](#).

The hardware architecture of Arm CCA is called the Realm Management Extension (RME), and is described in [Arm Architecture Reference Manual Supplement, The Realm Management Extension \(RME\), for Armv9-A \[2\]](#).

A1.1 Confidential computing

The Armv8-A architecture ([Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)) includes mechanisms that establish a privilege hierarchy. Software operating at higher privilege levels is responsible for managing the resources (principally memory and processor cycles) that are used by entities at lower privilege levels.

Prior to Arm CCA, resource management was coupled with a right of access. That is, a resource that is managed by a higher-privileged entity is also accessible by it. A *Realm* is a protected execution environment for which this coupling is broken, so that the right to manage resources is separated from the right to access those resources.

The purpose of a Realm is to provide to the Realm owner an environment for confidential computing, without requiring the Realm owner to trust the software components that manage the resources used by the Realm.

Construction of a Realm, and allocation of resources to a Realm at runtime, are the responsibility of the Virtual Machine Monitor (VMM). In this specification, the term *Host* is used to refer to the VMM.

See also:

- [A2.1 Realm](#)

A1.2 System software components

The system software architecture of Arm CCA is summarised in the following figure.

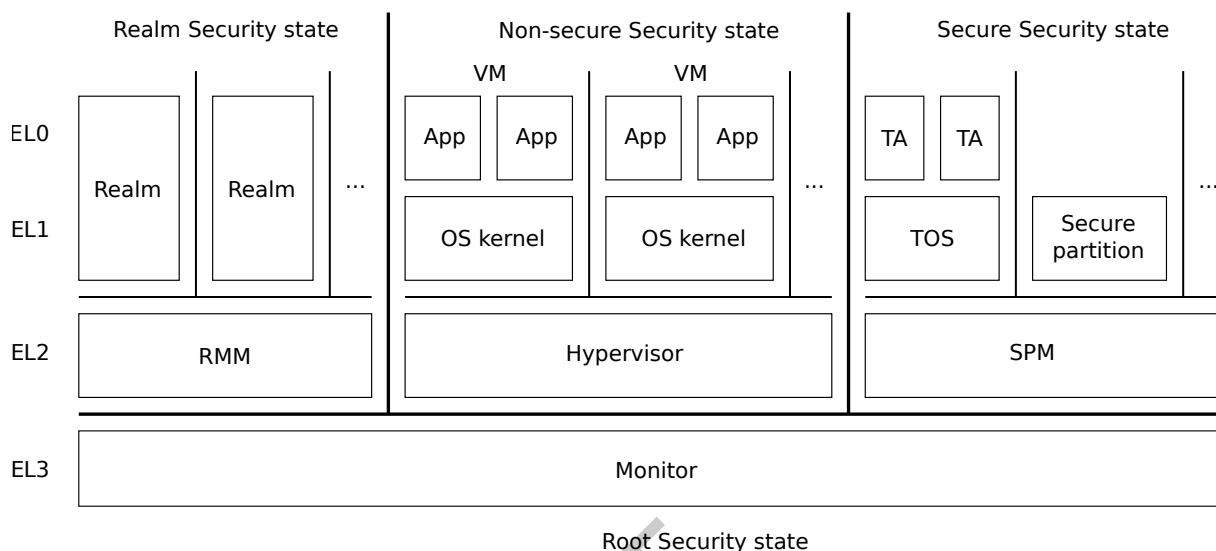


Figure A1.1: System software architecture

The components shown in the diagram are listed below.

Component	Description
Monitor	The most privileged software component, which is responsible for switching between the Security states used at EL2, EL1 and EL0.
Realm	A protected execution environment.
Realm Management Monitor (RMM)	The software component which is responsible for the management of Realms.
Virtual Machine (VM)	An execution environment within which an operating system can run. Note that a Realm is a VM which executes in the Realm security state.
Hypervisor	The software component which is responsible for the management of VMs.
Secure Partition Manager (SPM)	The software component which is responsible for the management of Secure Partitions.
Trusted OS (TOS)	An operating system which runs in a Secure Partition.
Trusted Application (TA)	An application hosted by a TOS.

A1.3 Realm Management Monitor

The Realm Management Monitor (RMM) is the system component that is responsible for the management of Realms.

The responsibilities of the RMM are to:

- Provide services that allow the Host to create, populate, execute and destroy Realms.
- Provide services that allow the initial configuration and contents of a Realm to be attested.
- Protect the confidentiality and integrity of Realm state during the lifetime of the Realm.
- Protect the confidentiality of Realm state during and following destruction of the Realm.
- Act as the Trusted Security Manager (TSM) in Realm device assignment.

The RMM exposes the following interfaces, which are accessed via SMC instructions, to the Host:

- The *Realm Management Interface* (RMI), which provides services for the creation, population, execution and destruction of Realms.

The RMM exposes the following interfaces, which are accessed via SMC instructions, to Realms:

- The *Realm Services Interface* (RSI), which provides services used to manage resources allocated to the Realm, and to request an attestation report.
- The *Power State Coordination Interface* (PSCI), which provides services used to control power states of VPEs within a Realm. Note that the HVC conduit for PSCI is not supported for Realms.

The RMM operates by manipulating data structures which are stored in memory accessible only to the RMM.

See also:

- [Chapter A9 Realm device assignment](#)
- [Chapter B4 Realm Management Interface](#)
- [Chapter B5 Realm Services Interface](#)
- [Chapter B6 Power State Control Interface](#)

Chapter A2

Concepts

This chapter introduces the following concepts which are central to the RMM architecture:

- [A2.1 Realm](#)
- [A2.2 Granule](#)
- [A2.3 Realm Execution Context](#)

A2.1 Realm

This section describes the concept of a Realm.

A2.1.1 Overview

D_DLRSR A *Realm* is an execution environment which is protected from agents in the Non-secure and Secure Security states, and from other Realms.

A2.1.2 Realm execution environment

I_LQYLY The execution environment of a Realm is an EL0 + EL1 environment, as described in [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#).

A2.1.2.1 Realm registers

R_NJHQK On first entry to a Realm VPE, PE state is initialized according to “PE state on reset to AArch64 state” in [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#), except for GPR and PC values which are specified by the Host during Realm creation.

G_ZFCQX Confidentiality is guaranteed for a Realm VPE’s general purpose and SIMD / floating point registers.

G_QHZCS Confidentiality is guaranteed for other Realm VPE register state (including stack pointer, program counter and EL0 / EL1 system registers).

G_XRMHP Integrity is guaranteed for a Realm VPE’s general purpose and SIMD / floating point registers.

G_YKRWG Integrity is guaranteed for other Realm VPE register state (including stack pointer, program counter and EL0 / EL1 system registers).

I_GPGFB A Realm can use a Host call to pass arguments to the Host and receive results from the Host.

See also:

- [A2.3 Realm Execution Context](#)
- [A4.5 Host call](#)
- [B4.3.25 RMI_REALM_CREATE command](#)

A2.1.2.2 Realm memory

I_TQMMZ A Realm is able to determine whether a given IPA is *protected* or *unprotected*.

G_LQFQH Confidentiality is guaranteed for memory contents accessed via a protected address. Informally, this means that a change to the contents of such a memory location is not observable by any agent outside the *CCA platform*.

G_QMLCJ Integrity is guaranteed for memory contents accessed via a protected address. Informally, this means that the Realm does not observe the contents of the location to change unless the Realm itself has either written a different value to the location, or provided consent to the RMM for integrity of the location to be violated.

See also:

- [A5.2.1 Realm IPA space](#)

A2.1.2.3 Realm processor features

R_JGHYJ The value returned to a Realm from reading a feature register is architecturally valid and describes the set of features which are present in the Realm’s execution environment.

I_KKBDP The RMM may suppress a feature which is supported by the underlying hardware platform, if exposing that feature to a Realm could lead to a security vulnerability.

See also:

- [Chapter A3 Feature discovery and configuration](#)

A2.1.2.4 IMPDEF system registers

R_{FQCKH} A Realm read from or write to an IMPLEMENTATION DEFINED system register causes an Unknown exception taken to the Realm.

A2.1.3 Realm attributes

This section describes the attributes of a Realm.

D_{JSGFY} A *Realm attribute* is a property of a Realm whose value can be observed or modified either by the Host or by the Realm.

I_{TTDVX} An example of a way in which a Realm attribute may be observable is the outcome of an RMM command.

D_{MHJCK} The attributes of a Realm are summarized in the following table.

Name	Type	Description
feat_lpa2	RmmFeature	Whether LPA2 is enabled for this Realm
ipa_width	UInt8	IPA width in bits
measurements	RmmRealmMeasurement [5]	Realm measurements
hash_algo	RmmHashAlgorithm	Algorithm used to compute Realm measurements
rec_index	UInt64	Index of next REC to be created
rtt_base	Address [4]	Realm Translation Table base addresses If rtt_tree_per_plane is FEATURE_FALSE then only the first entry is valid. If rtt_tree_per_plane is FEATURE_TRUE then only the first (num_aux_planes + 1) entries are valid.
rtt_level_start	Int64	RTT starting level
rtt_num_start	UInt64	Number of physically contiguous starting level RTTs
state	RmmRealmState	Lifecycle state
vmid	Bits 16[4]	Virtual Machine Identifiers If rtt_tree_per_plane is FEATURE_FALSE then only the first entry is valid. If rtt_tree_per_plane is FEATURE_TRUE then only the first (num_aux_planes + 1) entries are valid.
rpv	Bits 512	Realm Personalization Value
feat_da	RmmFeature	Whether Realm device assignment is enabled for this Realm
rtt_tree_per_plane	RmmFeature	Whether this Realm has an RTT tree per Plane
num_aux_planes	UInt64	Number of auxiliary Planes
rtt_s2ap_encoding	RmmRttS2APEncoding	S2AP encoding
overlay_perms	RmmMemPerms [4]	Memory overlay permissions
overlay_locked	RmmMemPermLocked [16]	Whether memory overlay value is locked
lfa_policy	RmmLfaPolicy	Live Firmware Activation policy for components within the Realm's TCB
mecid	Bits 64	Memory Encryption Context Identifier

Name	Type	Description
mec_policy	RmmMecPolicy	MEC policy
num_recs	UInt64	Number of RECs owned by this Realm
num_vdevs	UInt64	Number of VDEVs owned by this Realm
vdev_count	UInt64	Number of VDEVs which have been assigned to this Realm

D_{MGGPT} A *Realm Initial Measurement* (RIM) is a measurement of the configuration and contents of a Realm at the time of activation.

D_{GRFCS} A *Realm Extensible Measurement* (REM) is a measurement value which can be extended during the lifetime of a Realm.

I_{FMPYL} Attributes of a Realm include an array of measurement values. The first entry in this array is a RIM. The remaining entries in this array are REMs.

X_{DNDKV} During Realm creation, the Host provides ipa_width, rtt_level_start and rtt_num_start values as Realm parameters. According to the VMSA, the rtt_num_start value is architecturally defined as a function of the ipa_width and rtt_level_start values. It would therefore have been possible to design the Realm creation interface such that the Host provided only the ipa_width and rtt_level_start values. However, this would potentially allow a Realm to be successfully created, but with a configuration which did not match the Host's intent. For this reason, it was decided that the Host should specify all three values explicitly, and that Realm creation should fail if the values are not consistent. See [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#) for further details.

I_{QRVTT} The VMID of a Realm is chosen by the Host. The VMID must be within the range supported by the hardware platform. The RMM ensures that every Realm on the system has a unique VMID.

D_{FTWBK} A *Realm Personalization Value* (RPV) is a provided by the Host, to distinguish between Realms which have the same Realm Initial Measurement, but different behavior.

S_{FCNBF} Possible uses of the RPV include:

- A GUID
- Hash of Realm Owner public key
- Hash of a “personalisation document” which is provided to the Realm via a side-band (for example, via NS memory) and contains configuration information used by Realm software.

I_{ZFSWC} The RMM treats the RPV as an opaque value.

I_{BFSRK} The RPV is included in the Realm attestation report as a separate claim.

I_{MFRXD} The RPV is included in the output of the RSI_REALM_CONFIG command.

I₀₀₀₁ If Realm device assignment is not enabled for a Realm then all of the following are true:

- Assignment of a virtual device to the Realm by execution of RMI_VDEV_CREATE fails.
- The device assignment feature is reported to the Realm by RSI_FEATURES as not enabled. Consequently, execution of any RSI_RDEV command fails.

See also:

- [A2.1.5 Realm lifecycle](#)
- [A2.3 Realm Execution Context](#)
- [A3.3 Realm LPA2 and IPA width](#)
- [A5.2.1 Realm IPA space](#)
- [A5.6 Realm Translation Table](#)
- [A7.1 Realm measurements](#)
- [A7.2.3.1.3 Realm Personalization Value claim](#)

- [B4.3.52 RMI_VDEV_CREATE command](#)
- [B5.3.3 RSI_FEATURES command](#)
- [B5.3.25 RSI_REALM_CONFIG command](#)
- [C2.31 RmmRealm type](#)

A2.1.4 Realm liveness

D_WTXTJ *Realm liveness* is a property which means that there exists one or more Granules, other than the RD and the starting level RTTs, which are owned by the Realm.

I_PVPOB If a Realm is live, it cannot be destroyed.

D_PCKRN A Realm is *live* if any of the following is true:

- The number of RECs owned by the Realm is not zero
- A starting level RTT of the Realm is live
- The number of VDEVs owned by the Realm is not zero
- The number of VSMMUs owned by the Realm is not zero

I_VKKPJ If a Realm owns a non-zero number of Data Granules, this implies that it has a starting level RTT which is live, and therefore that the Realm itself is live.

See also:

- [A2.1.5 Realm lifecycle](#)
- [A2.2.3 Granule lifecycle](#)
- [A2.2.4 Granule ownership](#)
- [A2.3 Realm Execution Context](#)
- [A5.6.8 RTTE liveness and RTT liveness](#)
- [A9.4 Virtual device object](#)
- [A9.6 Virtual SMMU](#)
- [B3.62 RealmsLive function](#)
- [B4.3.26 RMI_REALM_DESTROY command](#)

A2.1.5 Realm lifecycle

See also:

- [Chapter A3 Feature discovery and configuration](#)
- [D1.2 Realm lifecycle flows](#)

A2.1.5.1 States

D_GDQPJ The states of a Realm are listed below.

State	Description
REALM_NEW	Under construction. Not eligible for execution.
REALM_ACTIVE	Eligible for execution.
REALM_SYSTEM_OFF	System has been turned off. Not eligible for execution.

A2.1.5.2 State transitions

I_RRRHG Permitted Realm state transitions are shown in the following table. The rightmost column lists the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of a Realm object. A transition to the pseudo-state *NULL* represents destruction of a Realm object.

From state	To state	Events
<i>NULL</i>	REALM_NEW	RMI_REALM_CREATE
REALM_NEW	<i>NULL</i>	RMI_REALM_DESTROY
REALM_ACTIVE	<i>NULL</i>	RMI_REALM_DESTROY
REALM_SYSTEM_OFF	<i>NULL</i>	RMI_REALM_DESTROY
REALM_NEW	REALM_ACTIVE	RMI_REALM_ACTIVATE
REALM_ACTIVE	REALM_SYSTEM_OFF	PSCI_SYSTEM_OFF PSCI_SYSTEM_RESET

I_{YCPWW}

Permitted Realm state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of an RD. A transition to the pseudo-state *NULL* represents destruction of an RD.

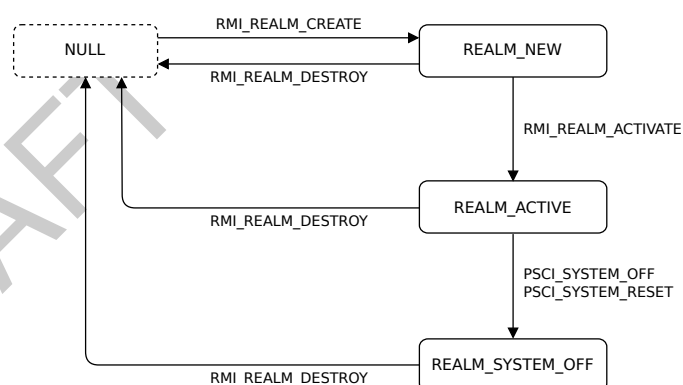


Figure A2.1: Realm state transitions

See also:

- [B6.3.6 PSCI_SYSTEM_OFF command](#)
- [B6.3.7 PSCI_SYSTEM_RESET command](#)
- [B4.3.24 RMI_REALM_ACTIVATE command](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.3.26 RMI_REALM_DESTROY command](#)

A2.1.6 Realm parameters

D_{TGMVZ}

A *Realm parameter* is a value which is provided by the Host during Realm creation.

See also:

- [A2.1.3 Realm attributes](#)
- [Chapter A3 Feature discovery and configuration](#)
- [B3.91 RmiRealmParamsAt function](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.4.29 RmiRealmParams type](#)

A2.1.7 Realm Descriptor

D_{TNSBY}

A *Realm Descriptor* (RD) is an RMM data structure which stores attributes of a Realm.

D_{GGKWX}

The size of an RD is one Granule.

See also:

- [A2.1.3 Realm attributes](#)
- [A2.2.3 Granule lifecycle](#)

DRAFT

A2.2 Granule

This section describes the concept of a Granule.

D_{NBXXX} A *Granule* is a unit of physical memory whose size is 4KB.

I_{DJGZW} A Granule may be used for one of the following purposes:

- To store code or data used by the Host
- To store code or data used by software in the Secure Security state
- To store code or data used by a Realm
- To access device memory, such as a memory-mapped register interface
- To store data used by the RMM to manage a Realm

The use of a Granule is reflected in its lifecycle state.

A2.2.1 Granule category

D_{ZVRXC} A Granule is *delegable* if it can be delegated by the Host for use by the RMM or by a Realm.

D₀₀₀₂ *Granule category* is a static property of each physical location in a system.

I₀₀₀₃ The category of a Granule determines:

- Whether the Granule can be used to store RMM data
- Whether the Granule can be allocated to a Realm, and if so what constraints are imposed on the attributes of mappings to the Granule.

D₀₀₀₄ The set of Granule categories are listed in the following table.

Granule category	Description
Delegable DRAM	DRAM which can be used to store RMM data, or allocated to a Realm
Delegable non-coherent device memory	Non-coherent device memory which can be allocated to a Realm
Delegable coherent device memory	Coherent device memory which can be allocated to a Realm
Non-delegable memory	Memory which cannot be used to store RMM data, or allocated to a Realm

D₀₀₀₅ *Delegable device memory* is the union of Delegable non-coherent device memory and Delegable coherent device memory.

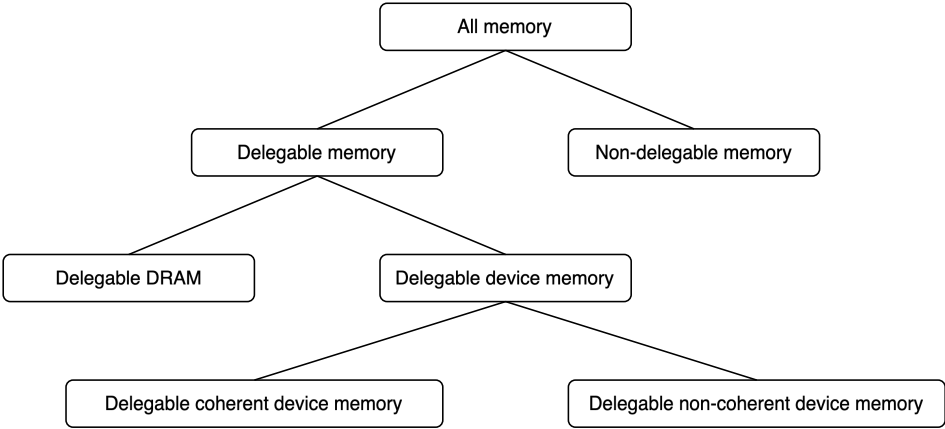
D₀₀₀₆ *Delegable memory* is the union of Delegable DRAM and Delegable device memory.

U_{KHKL P} In a typical implementation, all memory which is presented to the Host as RAM is Delegable DRAM.

Examples of Non-delegable memory may include memory which is carved out for use by the Root world, the RMM or the Secure world.

I₀₀₀₇

The following diagram summarizes the relationship between Granule categories.



See also:

- [Chapter A9 Realm device assignment](#)

A2.2.2 Granule attributes

This section describes the attributes of a Granule.

D_{JPBBC}

A *Granule attribute* is a property of a Granule whose value can be observed or modified either by the Host or by a Realm.

I_{WVXGK}

Examples of ways in which a Granule attribute may be observable include the outcome of an RMM command, and whether a memory access generates a fault.

D_{DVMRF}

The attributes of a Granule are summarized in the following table.

Name	Type	Description
gpt	RmmGptEntry	GPT entry
state	RmmGranuleState	Lifecycle state

See also:

- [A2.1 Realm](#)
- [A2.1.7 Realm Descriptor](#)
- [A2.2.3 Granule lifecycle](#)
- [B3.30 GranuleAccessPermitted function](#)
- [C2.12 RmmGranule type](#)

A2.2.3 Granule lifecycle

A2.2.3.1 States

D_{MPLGT}

The states of a Granule are listed below.

For each state, the corresponding GPT entry value is shown.

Granule state	Description	GPT entry
UNDELEGATED	Not delegated for use by the RMM.	Not GPT_REALM

Granule state	Description	GPT entry
DELEGATED	Delegated for use by the RMM.	GPT_REALM
RD	Realm Descriptor.	GPT_REALM
REC	Realm Execution Context.	GPT_REALM
REC_AUX	Realm Execution Context auxiliary Granule.	GPT_REALM
DATA	Realm code or data.	GPT_REALM
RTT	Realm Translation Table.	GPT_REALM
PDEV	Physical device.	GPT_REALM
PDEV_AUX	Physical device auxiliary Granule.	GPT_REALM
VDEV	Virtual device.	GPT_REALM
DEV_MAPPED	Device memory, mapped into a Realm.	GPT_REALM
VSMMU	Virtual SMMU.	GPT_REALM

I_{MPGJV} If the state of a Granule is UNDELEGATED then the RMM does not prevent the GPT entry of the Granule from being changed by another agent to any value except GPT_REALM.

D_{VRSKZ} An NS Granule is a Granule whose GPT entry is GPT_NS.

A2.2.3.2 State transitions

I₀₀₀₈ The initial state of all Granules of Delegable memory is UNDELEGATED.

I₀₀₀₉ The set of reachable states depends on the Granule category.

I_{2JBT} Permitted Granule state transitions are shown in the following table. The rightmost column lists the events which can cause the corresponding state transition.

From state	To state	Events
UNDELEGATED	DELEGATED	RMI_GRANULE_DELEGATE
DELEGATED	UNDELEGATED	RMI_GRANULE_UNDELEGATE
DELEGATED	RD	RMI_REALM_CREATE
RD	DELEGATED	RMI_REALM_DESTROY
DELEGATED	DATA	RMI_DATA_CREATE RMI_DATA_CREATE_UNKNOWN
DATA	DELEGATED	RMI_DATA_DESTROY
DELEGATED	REC	RMI_REC_CREATE
REC	DELEGATED	RMI_REC_DESTROY
DELEGATED	REC_AUX	RMI_REC_CREATE
REC_AUX	DELEGATED	RMI_REC_DESTROY
DELEGATED	RTT	RMI_REALM_CREATE RMI_RTT_CREATE

From state	To state	Events
RTT	DELEGATED	RMI_REALM_DESTROY RMI_RTT_DESTROY
DELEGATED	PDEV	RMI_PDEV_CREATE
PDEV	DELEGATED	RMI_PDEV_DESTROY
DELEGATED	PDEV_AUX	RMI_PDEV_CREATE
PDEV_AUX	DELEGATED	RMI_PDEV_DESTROY
DELEGATED	VDEV	RMI_VDEV_CREATE
VDEV	DELEGATED	RMI_VDEV_DESTROY
DELEGATED	DEV_MAPPED	RMI_DEV_MEM_MAP
DEV_MAPPED	DELEGATED	RMI_DEV_MEM_UNMAP
DELEGATED	VSMMU	RMI_VSMMU_CREATE
VSMMU	DELEGATED	RMI_VSMMU_DESTROY

I_{VVGVM}

Permitted Granule state transitions are shown in the following figures. Each arc is labeled with the events which can cause the corresponding state transition.

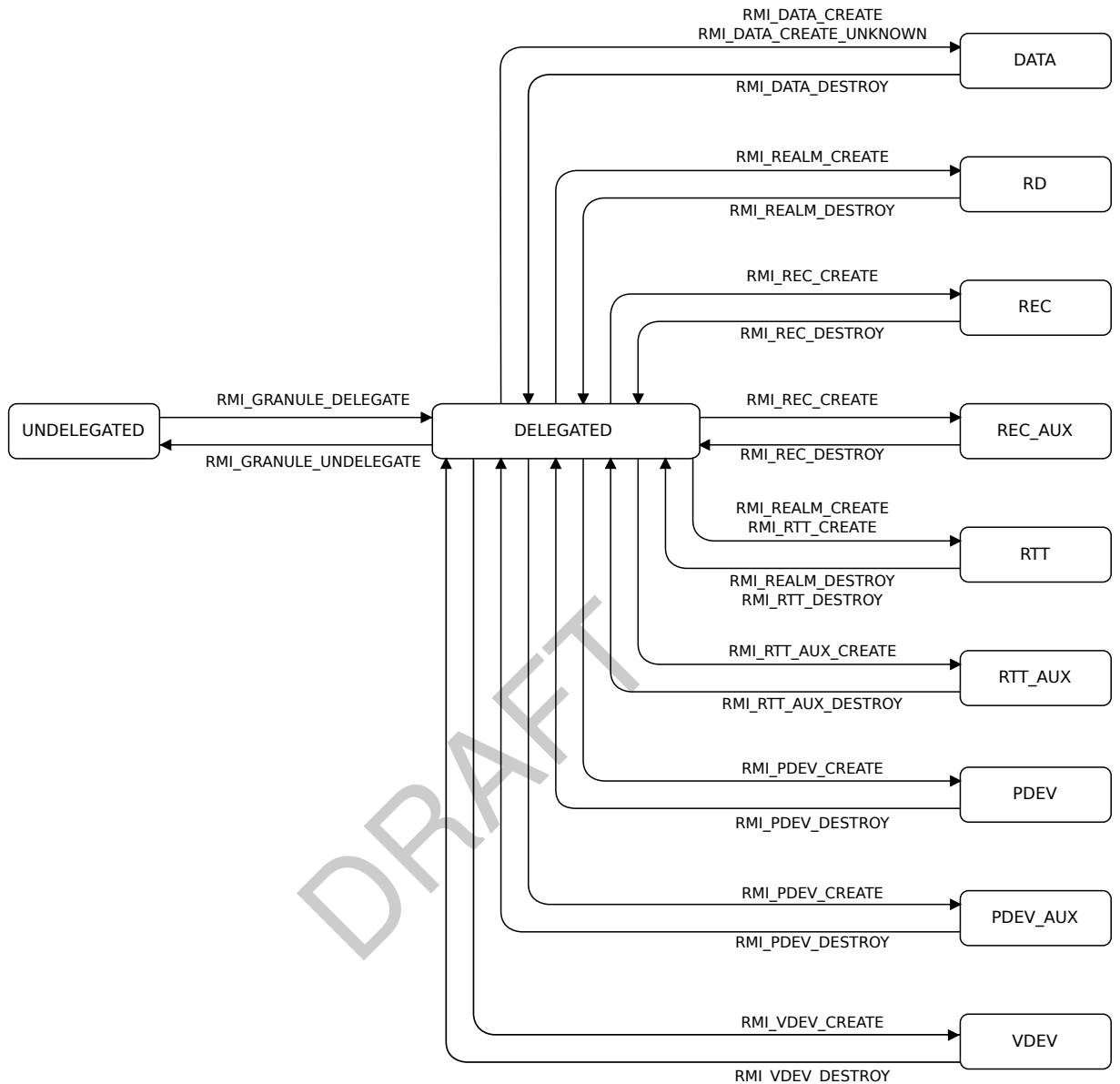


Figure A2.2: Granule state transitions for Delegable DRAM



Figure A2.3: Granule state transitions for Delegable device memory

See also:

- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.4 RMI_DEV_MEM_MAP command](#)
- [B4.3.5 RMI_DEV_MEM_UNMAP command](#)
- [B4.3.7 RMI_GRANULE_DELEGATE command](#)

- [B4.3.8 RMI_GRANULE_UNDELEGATE command](#)
- [B4.3.14 RMI_PDEV_CREATE command](#)
- [B4.3.15 RMI_PDEV_DESTROY command](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.3.26 RMI_REALM_DESTROY command](#)
- [B4.3.28 RMI_REC_CREATE command](#)
- [B4.3.29 RMI_REC_DESTROY command](#)
- [B4.3.38 RMI_RTT_CREATE command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)
- [B4.3.52 RMI_VDEV_CREATE command](#)
- [B4.3.53 RMI_VDEV_DESTROY command](#)
- [B4.3.57 RMI_VSMMU_CREATE command](#)
- [B4.3.58 RMI_VSMMU_DESTROY command](#)

A2.2.4 Granule ownership

I_{DMVQM}

A Granule whose state is none of the following is owned by a Realm:

- UNDELEGATED
- DELEGATED
- PDEV
- DEV_MAPPED

I_{PRNTM}

The owner of a Granule is identified by the address of a Realm Descriptor (RD).

I_{ZXBZM}

For a Granule whose state is RD, the ownership relation is recursive: the owning Realm is identified by the address of the RD itself.

I_{TYHTD}

A Granule whose state is RTT is one of the following:

- A starting level RTT. The address of this RTT is stored in the RD of the owning Realm.
- A non-starting level RTT. The address of this RTT is stored in its parent RTT, in an RTT entry whose state is TABLE. Recursively following the parent relationship leads to the RD of the owning Realm.

I_{QCNRM}

A Granule whose state is DATA is mapped at a Protected IPA, in an RTT entry whose state is ASSIGNED. The Realm which owns the RTT is the owner of the DATA Granule.

I_{HHPVB}

A REC has an “owner” attribute which points to the RD of the owning Realm.

X_{NDNHG}

A REC is not mapped at a Protected IPA. Its ownership therefore needs to be recorded explicitly.

I₀₀₁₀

A VDEV has an “owner” attribute which points to the RD of the owning Realm.

X₀₀₁₁

A VDEV is not mapped at a Protected IPA. Its ownership therefore needs to be recorded explicitly.

See also:

- [A2.1 Realm](#)
- [A2.1.7 Realm Descriptor](#)
- [A2.3 Realm Execution Context](#)
- [A5.2.1 Realm IPA space](#)
- [A5.6 Realm Translation Table](#)
- [Chapter A9 Realm device assignment](#)
- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.28 RMI_REC_CREATE command](#)
- [B4.3.38 RMI_RTT_CREATE command](#)

A2.2.5 Granule wiping

R _{TMGSL}	When the state of a Granule has transitioned from <i>P</i> to DELEGATED and then to any other state, any content associated with <i>P</i> has been <i>wiped</i> .
X _{CTGQZ}	Any sequence of Granule state transitions which passes through the DELEGATED state causes the Granule contents to be wiped. This is necessary to ensure that information does not leak from one Realm to another, or from a Realm to the Host. Note that no agent can observe the contents of a Granule while its state is DELEGATED.
D _{WTWJR}	<i>Wiping</i> is an operation which changes the observable value of a memory location from <i>X</i> to <i>Y</i> , such that the value <i>X</i> cannot be determined from the value <i>Y</i> .
R _{BSXXV}	Wiping of a memory location does not reveal, directly or indirectly, any confidential Realm data.
I _{MRPCQ}	Wiping is not guaranteed to be implemented as zero filling.
S _{VJWYH}	Realm software should not assume that the initial contents of uninitialized memory (that is, Realm IPA space which is backed by DATA Granules created using RMI_DATA_CREATE_UNKNOWN) are zero.

See also:

- [Arm CCA Security model \[4\]](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)

DRAFT

A2.3 Realm Execution Context

This section describes the concept of a Realm Execution Context (REC).

A2.3.1 Overview

D_{LRFCP}

A *Realm Execution Context* (REC) is an R-EL0&1 execution context which is associated with a Realm VPE.

A *REC object* is an RMM data structure which is used to store the register state of a REC.

See also:

- [A2.1.2 Realm execution environment](#)
- [Chapter A4 Realm exception model](#)

A2.3.2 REC attributes

This section describes the attributes of a REC.

D_{ZLGLT}

A *REC attribute* is a property of a REC whose value can be observed or modified either by the Host or by the Realm which owns the REC.

I_{CSGGT}

Examples of ways in which a REC attribute may be observable include the outcome of an RMM command, and the PE state following Realm entry.

D_{LQSFT}

The attributes of a REC are summarized in the following table.

Name	Type	Description
owner	Address	PA of RD of Realm which owns this REC
aux	Address [16]	PA's of auxiliary Granules
flags	RmmRecFlags	Flags which control REC behavior
mpidr	Bits64	MPIDR value
gic_owner	UInt64	Index of Plane which is the GIC owner
state	RmmRecState	Lifecycle state
pending	RmmRecPending	Whether a REC operation is pending
emulatable_abort	RmmRecEmulatableAbort	Whether the most recent exit from this REC was due to an Emulatable Data Abort
gprs	Bits64 [32]	General-purpose register values
pc	Bits64	Program counter value
sysregs	RmmSystemRegisters	EL1 and EL0 system register values
attest_state	RmmRecAttestState	Attestation token generation state
attest_challenge	Bits512	Challenge for under-construction attestation token
ripas_addr	Address	Next IPA to be processed in RIPAS change
ripas_top	Address	Top IPA of pending RIPAS change
ripas_value	RmmRipas	RIPAS value of pending RIPAS change
ripas_destroyed	RmmRipasChangeDestroyed	Whether a RIPAS change from DESTROYED should be permitted
ripas_response	RmmRecResponse	Host response to RIPAS change request

Name	Type	Description
dev_mem_addr	Address	Next IPA to be processed in device memory mapping validation
dev_mem_top	Address	Top IPA of pending device memory mapping validation
dev_mem_pa	Address	PA of device memory
dev_mem_flags	RmmDevMemFlags	Device memory mapping validation flags
dev_mem_response	RmmRecResponse	Host response to device memory mapping validation request
s2ap_addr	Address	Next IPA to be processed in S2AP change
s2ap_top	Address	Top IPA of pending S2AP change
s2ap_overlay_index	UInt4	Overlay index of pending S2AP change
s2ap_response	RmmRecResponse	Host response to S2AP change request
vdev_id	Bits64	Virtual device ID
inst_id	UInt64	Device instance ID
inst_id_valid	RmmBoolean	Whether device instance ID is valid

I_PVMTY	The <i>aux</i> attribute of a REC is a list of <i>auxiliary Granules</i> .
I_RWFZF	The number of auxiliary Granules required for a REC is returned by the RMI_REC_AUX_COUNT command.
X_LRWHB	Depending on the configuration of the CCA platform and of the Realm, the amount of storage space required for a REC may exceed a single Granule.
I_TGLBK	The number of auxiliary Granules required for a REC can vary between Realms on a CCA platform.
R_MMBNR	The number of auxiliary Granules required for a REC is a constant for the lifetime of a given Realm.
I_BGVRT	The <i>gprs</i> attribute of a REC is the set of general-purpose register values which are saved by the RMM on exit from the REC and restored by the RMM on entry to the REC.
I_FPJDL	The <i>mpidr</i> attribute of a REC is a value which can be used to identify the VPE associated with the REC.
I_BLVKZ	The <i>pc</i> attribute of a REC is the program counter which is saved by the RMM on exit from the REC and restored by the RMM on entry to the REC.
I_GHFNQ	The <i>runnable</i> flag of a REC determines whether the REC is eligible for execution. The RMI_REC_ENTER command results in a REC entry only if the value of the flag is RUNNABLE.
I_SCCMH	The runnable flag of a REC is controlled by the Realm. Its initial value is reflected in the Realm Initial Measurement, and during Realm execution its value can be changed by execution of the PSCI_CPU_ON and PSCI_CPU_OFF commands.
I_PMYBG	The <i>state</i> attribute of a REC is controlled by the Host, by execution of the RMI_REC_ENTER command.
D_CDXDZ	The <i>sysregs</i> attribute of a REC is the set of system register values which are saved by the RMM on exit from the REC and restored by the RMM on entry to the REC.
D_0012	The <i>gic_owner</i> attribute of a REC is the index of the Plane which is the GIC owner for the REC.

See also:

- [A2.3.3 REC index and MPIDR value](#)
- [A2.3.4 REC lifecycle](#)
- [A4.3.4.3 REC exit due to Data Abort](#)

- [A10.4 Planes interrupts](#)
- [B4.3.30 RMI_REC_ENTER command](#)
- [B6.3.2 PSCI_CPU_OFF command](#)
- [B6.3.3 PSCI_CPU_ON command](#)
- [C2.34 RmmRec type](#)

A2.3.3 REC index and MPIDR value

D_{KQVHN}

The *REC index* is the unsigned integer value generated by concatenation of MPIDR fields:

$\text{index} = \text{Aff3}:\text{Aff2}:\text{Aff1}:\text{Aff0}[3:0]$

This is illustrated by the following table.

REC index	Aff3	Aff2	Aff1	Aff0[3:0]
0	0	0	0	0
1	0	0	0	1
...
16	0	0	1	0
...
4096	0	1	0	0
...
1048576	1	0	0	0
...

I_{PVLZY}

The $\text{Aff0}[7:4]$ field of a REC MPIDR value is RES0 for compatibility with GICv3.

I_{TTWVM}

When creating the n th REC in a Realm, the Host is required to use the MPIDR corresponding to REC index n .

See also:

- [B3.70 RecIndex function](#)
- [B4.3.28 RMI_REC_CREATE command](#)
- [B4.4.35 RmiRecMpidr type](#)

A2.3.4 REC lifecycle

A2.3.4.1 States

D_{HTXQY}

The states of a REC are listed below.

State	Description
REC_READY	REC is not currently running.
REC_RUNNING	REC is currently running.

A2.3.4.2 State transitions

I_{PHMWT}

Permitted REC state transitions are shown in the following table. The rightmost column lists the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of a REC object. A transition to the pseudo-state *NULL* represents destruction of a REC object.

From state	To state	Events
<i>NULL</i>	REC_READY	RMI_REC_CREATE
REC_READY	<i>NULL</i>	RMI_REC_DESTROY
REC_READY	REC_RUNNING	RMI_REC_ENTER
REC_RUNNING	REC_READY	Return from RMI_REC_ENTER

I_{FNSTJ}

Permitted REC state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of a REC. A transition to the pseudo-state *NULL* represents destruction of a REC.

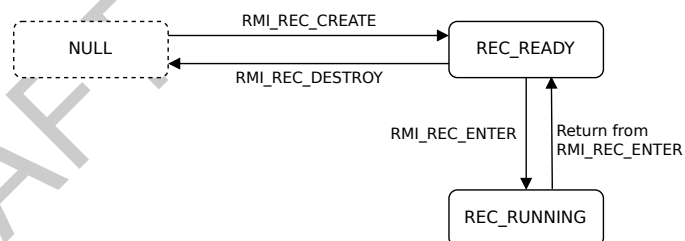


Figure A2.4: REC state transitions

I_{LYXCN}

The maximum number of RECs per Realm is an IMPLEMENTATION DEFINED value which is discoverable via [RMI_FEATURES](#).

See also:

- [B4.3.28 RMI_REC_CREATE command](#)
- [B4.3.29 RMI_REC_DESTROY command](#)
- [B4.3.30 RMI_REC_ENTER command](#)

See also:

- [B4.3.6 RMI_FEATURES command](#)

Chapter A3

Feature discovery and configuration

This section describes how the Host discovers features which are supported by the RMM implementation, and how the Host configures the features which are used by or available to a Realm.

A3.1 Feature discovery and configuration overview

I _{GJSMC}	RMM implementations across different CCA platforms may support disparate features and may offer disparate configuration options for Realms.
I _{YRSDX}	The features supported by an RMI implementation are discovered by reading feature pseudo-register values using the RMI_FEATURES command.
X _{WPHWG}	The term <i>pseudo-register</i> is used because, although these values are stored in memory, their usage model is similar to feature registers specified in the Arm A-profile architecture.
I _{QNJTQ}	On Realm creation, the Host provides a desired configuration in a Realm parameters structure to the RMI_REALM_CREATE command. The RMM checks that the configuration provided by the Host is supported by the implementation.
I _{RRHJJ}	Aspects of the Realm configuration which affect the security posture of the Realm are included in the Realm Initial Measurement.
I _{ZHXGX}	The features supported by an RSI implementation are discovered by reading feature pseudo-register values using the RSI_FEATURES command.

See also:

- [A2.1.6 Realm parameters](#)
- [A7.1.1 Realm Initial Measurement](#)
- [B3.32 ImplFeatures function](#)
- [B3.63 RealmParamsSupported function](#)
- [B4.3.6 RMI_FEATURES command](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B5.3.3 RSI_FEATURES command](#)

A3.2 Realm hash algorithm

I _{WMKGX}	The set of hash algorithms supported by the implementation is reported by the RMI_FEATURES command in RmiFeatureRegister0.
I ₀₀₁₃	The hash algorithm used by a Realm is provided by the Host when calling RMI_REALM_CREATE.
R _{KPBQM}	Providing an unsupported hash algorithm causes execution of RMI_REALM_CREATE to fail.

See also:

- [A7.1 Realm measurements](#)
- [B3.63 RealmParamsSupported function](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.4.14 RmiFeatureRegister0 type](#)

A3.3 Realm LPA2 and IPA width

I _{GVJMZ}	Support by the implementation for LPA2 is reported by the RMI_FEATURES command in RmiFeatureRegister0.
I _{NKLXQ}	Usage of LPA2 for Realm Translation Tables is configured by the Host when calling RMI_REALM_CREATE.
I _{LKJGN}	Realm IPA width is provided by the Host when calling RMI_REALM_CREATE.
R _{SZVDK}	Providing an unsupported IPA width (for example, smaller than the minimum supported, or larger than the maximum supported) causes execution of RMI_REALM_CREATE to fail.
I _{GKCCS}	The Host can choose a smaller IPA width than the maximum supported IPA width reported by RMI_FEATURES. This is true regardless of whether LPA2 is enabled for the Realm.

X _{FTVXQ}	<p>The Host may want to enable LPA2 for a Realm due to either or both of the following reasons:</p> <ul style="list-style-type: none"> • to allow the Realm to be configured with a larger IPA width • to allow access from mappings in the Realm's stage 2 translation to a larger PA space
I _{XDBQB}	A Realm can query its IPA width using the RSI_REALM_CONFIG command.
I _{FSNMG}	<p>If LPA2 is not enabled for a Realm then passing a PA greater than or equal to 2^{48} to any of the following commands causes an error to be returned:</p> <ul style="list-style-type: none"> • RMI_DATA_CREATE • RMI_DATA_CREATE_UNKNOWN • RMI_RTT_CREATE • RMI_RTT_AUX_CREATE • RMI_RTT_MAP_UNPROTECTED <p>See also:</p> <ul style="list-style-type: none"> • A5.2.1 Realm IPA space • B3.63 RealmParamsSupported function • B4.3.25 RMI_REALM_CREATE command • B4.4.14 RmiFeatureRegister0 type • B5.3.25 RSI_REALM_CONFIG command

A3.4 Realm support for Scalable Vector Extension

I _{KJVLJ}	Support by the implementation for the Scalable Vector Extension (FEAT_SVE) is reported by the RMI_FEATURES command in RmiFeatureRegister0.
I _{ZJSMJ}	Availability of SVE to a Realm is configured by the Host when calling RMI_REALM_CREATE.
I _{VNLNH}	SVE vector length for a Realm is provided by the Host when calling RMI_REALM_CREATE.
R _{FZZDS}	Providing a larger-than-supported SVE vector length causes execution of RMI_REALM_CREATE to fail. This is different from the behaviour of the hardware architecture, in which a larger-than-supported SVE vector length value is silently truncated.
X _{YGWTK}	The RMI ABI provides a natural mechanism to signal an invalid feature selection, via the return code of RMI_REALM_CREATE. The analog in the hardware architecture would be to generate an illegal exception return, which would cause undesirable coupling between two disparate parts of the architecture, namely the exception model and the SVE feature.
X _{CWNQC}	Providing a larger-than-supported SVE vector length causes execution of RMI_REALM_CREATE to fail prepares the architecture for addition of Realm live migration support in future. Assuming that the live migration flow starts with creation of an empty destination Realm, configured identically to the source Realm, this provides a point where the necessary feature support can be checked on the destination platform.
R _{NBYKC}	If SVE is supported by the platform but is disabled for the Realm via the RMI_REALM_CREATE command then a read of ID_AA64PFR0_EL1.SVE indicates that SVE is not supported.
U _{ZRJXL}	The RMM should trap and emulate reads of ID_AA64PFR0_EL1.SVE.
S _{VXRNN}	A Realm should discover SVE support by reading ID_AA64PFR0_EL1.SVE rather than based on the platform identity read from MIDR_EL1.

See also:

- [B3.63 RealmParamsSupported function](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.4.14 RmiFeatureRegister0 type](#)

A3.5 Realm support for self-hosted debug

I _{SSTJD}	Self-hosted debug is always available in Armv8-A.
I _{LVMFG}	The number of breakpoints and watchpoints are provided by the Host when calling RMI_REALM_CREATE.
R _{CJQTB}	Providing a number of breakpoints which is larger than the number of breakpoints available causes execution of RMI_REALM_CREATE to fail.
R _{PLMDH}	Providing a number of watchpoints which is larger than the number of watchpoints available causes execution of RMI_REALM_CREATE to fail.
X _{TPBHD}	Specifying that a larger-than-supported number of breakpoints or watchpoints causes execution of RMI_REALM_CREATE to fail prepares the architecture for addition of Realm live migration support in future. Assuming that the live migration flow starts with creation of an empty destination Realm, configured identically to the source Realm, this provides a point where the necessary feature support can be checked on the destination platform.

See also:

- [B3.63 RealmParamsSupported function](#)
- [B4.3.25 RMI_REALM_CREATE command](#)

A3.6 Realm support for Performance Monitors Extension

I _{RVCQD}	Support by the implementation for the Performance Monitors Extension (FEAT_PMU) is reported by the RMI_FEATURES command in RmiFeatureRegister0.
I _{NHCFD}	Availability of PMU to a Realm is configured by the Host when calling RMI_REALM_CREATE.
I _{XZMKC}	The number of PMU counters available to a Realm is provided by the Host when calling RMI_REALM_CREATE.
R _{XVRGD}	Providing a number of PMU counters which is larger than the number of PMU counters available causes RMI_REALM_CREATE to fail.
X _{NTWKF}	Specifying that a larger-than-supported number of PMU counters causes RMI_REALM_CREATE to fail prepares the architecture for addition of Realm live migration support in future. Assuming that the live migration flow starts with creation of an empty destination Realm, configured identically to the source Realm, this provides a point where the necessary feature support can be checked on the destination platform.

See also:

- [A8.1 Realm PMU](#)
- [B3.63 RealmParamsSupported function](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.4.14 RmiFeatureRegister0 type](#)

A3.7 Realm support for Activity Monitors Extension

R _{JJVZS}	The Activity Monitors Extension (FEAT_AMUv1) is not available to a Realm.
--------------------	---

A3.8 Realm support for Statistical Profiling Extension

R _{DCBNL}	The Statistical Profiling Extension (FEAT_SPE) is not available to a Realm.
--------------------	---

A3.9 Realm support for Trace Buffer Extension

R_{NXD}XG The Trace Buffer Extension (FEAT_TRBE) is not available to a Realm.

A3.10 Support for Realm device assignment

I₀₀₁₄ Support by the implementation for Realm device assignment is reported by the RMI_FEATURES command in RmiFeatureRegister0.

I₀₀₁₅ Availability of Realm device assignment for a Realm is configured by the Host when calling RMI_REALM_CREATE.
See also:

- [Chapter A9 Realm device assignment](#)
- [B3.63 RealmParamsSupported function](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.4.14 RmiFeatureRegister0 type](#)

A3.11 Support for auxiliary Planes

I₀₀₁₆ The maximum number of auxiliary Planes supported by the implementation is reported by the RMI_FEATURES command in the NUM_AUX_PLANES field of RmiFeatureRegister0.

R₀₀₁₇ The maximum number of auxiliary Planes supported by the implementation is either 0 or 3.

I₀₀₁₈ The number of auxiliary Planes for a Realm is provided by the Host when calling RMI_REALM_CREATE.

R₀₀₁₉ Providing a number of auxiliary Planes which is larger than the maximum number of auxiliary Planes causes RMI_REALM_CREATE to fail.

I₀₀₂₀ For a Realm with a non-zero number of auxiliary Planes, the RTT_PLANE field of RmiFeatureRegister0 indicates which one of the following configurations is supported by the implementation:

- The Realm has an RTT tree per Plane
- The Realm has a single RTT tree
- The Realm can be configured to either have an RTT tree per Plane, or a single RTT tree.

I₀₀₂₁ Whether a Realm has an RTT tree per Plane is configured by the Host when calling RMI_REALM_CREATE.

See also:

- [Chapter A10 Planes](#)
- [B3.32 ImplFeatures function](#)
- [B3.63 RealmParamsSupported function](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.4.14 RmiFeatureRegister0 type](#)

A3.12 Support for Stage 2 Access Permissions indirect encoding

I₀₀₂₂ The RTT_S2AP_INDIRECT field of RmiFeatureRegister0 indicates whether the Stage 2 Access Permissions (S2AP) value for a Realm IPA is:

- Encoded directly in the RTT entry, or
- Encoded indirectly, as described in “Stage 2 Indirect permissions” in [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#).

I₀₀₂₃ Whether a Realm uses S2AP indirect encoding is configured by the Host when calling RMI_REALM_CREATE.

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)
- [A5.6.11 Stage 2 Access Permissions](#)

- [B3.32 ImplFeatures function](#)
- [B3.63 RealmParamsSupported function](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.4.14 RmiFeatureRegister0 type](#)

A3.13 Live Firmware Activation

- I₀₀₂₄ Live Firmware Activation (LFA) allows an update to a platform firmware component to be activated without rebooting the system. This potentially includes components which are within the TCB of a Realm.
- I₀₀₂₅ A Realm has an LFA policy which is provided by the Host when calling RMI_REALM_CREATE.
- R₀₀₂₆ If the LFA policy of a Realm is LFA_DISALLOW then all firmware components within the Realm's TCB are guaranteed not to be live activated during the lifetime of the Realm.
- I₀₀₂₇ In order to apply LFA to any firmware component (including the RMM) which is within the TCB of a Realm whose LFA policy is LFA_DISALLOW, the Host must first destroy the Realm.
- I₀₀₂₈ The mechanism via which the LFA implementation determines whether any Realm with an LFA policy of LFA_DISALLOW currently exists on the system is IMPLEMENTATION DEFINED.
- I₀₀₂₉ If the LFA policy of a Realm is LFA_DISALLOW then the contents of the CCA platform software components claim reflect the state of all firmware components within the Realm's TCB, throughout the lifetime of the Realm.
- I₀₀₃₀ The LFA policy of a Realm is reflected in the Realm attestation token.
- See also:
- [Live Firmware Activation SMC Interface \[5\]](#)
 - [A7.2.3.2.7 CCA platform software components claim](#)
 - [B4.3.25 RMI_REALM_CREATE command](#)
 - [B4.4.19 RmiLfaPolicy type](#)

A3.14 GICv3 virtualization

- I₀₀₃₁ The number of GICv3 List Registers which can be provided by the Host via the RMI_REC_ENTER command is reported by the RMI_FEATURES command in RmiFeatureRegister0.
- X₀₀₃₂ Making the number of GICv3 List Registers discoverable via RMI allows the RMM to reserve List Registers for its own usage.
- See also:
- [A6.1 Realm interrupts](#)
 - [B4.3.30 RMI_REC_ENTER command](#)
 - [B4.4.14 RmiFeatureRegister0 type](#)

A3.15 Support for Realm memory encryption

- I₀₀₃₃ A Realm is configured on creation with a MEC policy.
- A MEC policy describes whether the Realm's memory encryption context is:
- Shared with other Realms
 - Private to the Realm
- See also:
- [A7.2.3.1 Realm claims](#)
 - [Chapter A11 Realm memory encryption](#)

Chapter A4

Realm exception model

This section describes how Realms are executed, and how exceptions which cause exit from a Realm are handled.

See also:

- [A2.1.2 Realm execution environment](#)

A4.1 Realm exception model overview

D_{HCGWL}	A <i>Realm entry</i> is a transfer of control to a Realm.
D_{RMGWJ}	A <i>Realm exit</i> is a transition of control from a Realm.
I_{SMPWB}	When executing in a Realm, an exception taken to R-EL2 or EL3 results in a Realm exit.
D_{XSNZP}	A <i>REC entry</i> is a Realm entry due to execution of RMI_REC_ENTER.
I_{FQZJG}	The Host provides the address of a REC as an input to the RMI_REC_ENTER command.
I_{MDQWG}	In this chapter, both <code>rec</code> and “the target REC” refer to the REC object which is provided to the RMI_REC_ENTER command.
D_{BLJGY}	A <i>RecRun object</i> is a data structure used to pass values between the RMM and the Host on REC entry and on REC exit.
I_{VCCFV}	A RecRun object is stored in Non-secure memory.
I_{WBHYZ}	The Host provides the address of a RecRun object as an input to the RMI_REC_ENTER command.
I_{HMSQM}	An implementation is permitted to return RMI_SUCCESS from RMI_REC_ENTER without performing a REC entry. For example, on observing a pending interrupt, the implementation can generate a REC exit due to IRQ without entering the target REC.
D_{TJCGH}	A <i>REC exit</i> is return from an execution of RMI_REC_ENTER which caused a REC entry.
I_{HPWVY}	The following diagram summarises the possible control flows that result from a Realm exit.

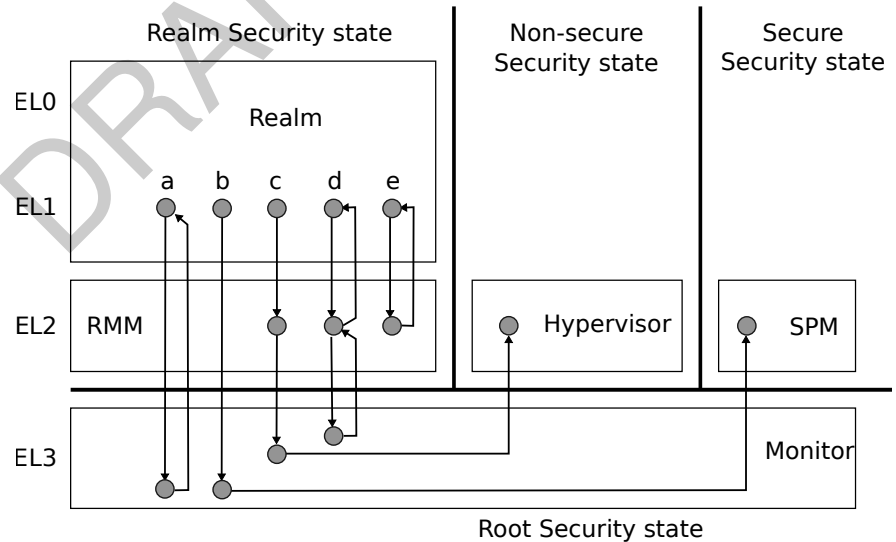


Figure A4.1: Realm exit paths

- The exception is taken to EL3. The Monitor handles the exception and returns control to the Realm.
- The exception is taken to EL3. The Monitor pre-empt's Realm Security state and passes control to the Secure Security state. This may be for example due to an FIQ.
- The exception is taken to EL2. The RMM decides to perform a REC exit. The RMM executes an SMC instruction, requesting the Monitor to pass control to the Non-secure Security state.
- The exception is taken to EL2. The RMM executes an SMC instruction, requesting the Monitor to perform an operation, then returns control to the Realm.
- The exception is taken to EL2. The RMM handles the exception and returns control to the Realm.

See also:

- [A4.2 REC entry](#)
- [A4.3 REC exit](#)
- [A10.2 Planes exception model](#)
- [B4.3.30 RMI_REC_ENTER command](#)
- [B4.4.37 RmiRecRun type](#)

DRAFT

A4.2 REC entry

This section describes REC entry.

See also:

- [A4.3 REC exit](#)
- [B4.3.30 RMI_REC_ENTER command](#)

A4.2.1 RmiRecEnter object

D _{SVSJM}	An <i>RmiRecEnter</i> object is a data structure used to pass values from the Host to the RMM on REC entry.
I _{YSKDN}	An <i>RmiRecEnter</i> object is stored in the <i>RecRun</i> object which is passed by the Host as an input to the <i>RMI_REC_ENTER</i> command.
I _{TRKKX}	On REC entry, execution state is restored from the REC object and from the <i>RmiRecEnter</i> object to the PE.
I _{GHDLM}	An <i>RmiRecEnter</i> object contains attributes which are used to manage Realm virtual interrupts.
D _{CLNLW}	The attributes of an <i>RmiRecEnter</i> object are summarized in the following table.

Name	Byte offset	Type	Description
flags	0x0	RmiRecEnterFlags	Flags
gprs[31]	0x200	Bits64	Registers
gicv3_hcr	0x300	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0x308	Bits64	GICv3 List Register values

I_{ZWRQP} In this chapter, both *rec_enter* and “the *RmiRecEnter* object” refer to the *RmiRecEnter* object which is provided to the *RMI_REC_ENTER* command.

I_{LFYDV} On REC entry, all *rec_enter* fields are ignored unless specified otherwise.

See also:

- [A2.3 Realm Execution Context](#)
- [A4.3.1 RmiRecExit object](#)
- [Chapter A6 Realm interrupts and timers](#)
- [B4.4.31 RmiRecEnter type](#)

A4.2.2 General purpose registers restored on REC entry

R _{NMSFT}	On REC entry, if the most recent exit from the target REC was a REC exit due to PSCI, then all of the following occur: <ul style="list-style-type: none">• X0 to X6 contain the PSCI return code and PSCI output values.• GPR values X7 to X30 are restored from the REC object to the PE.
R _{RZRRM}	On REC entry, if either this is the first entry to this REC, or the most recent exit from the target REC was not a REC exit due to PSCI, then GPR values X0 to X30 are restored from the REC object to the PE.
R _{YSNYQ}	On REC entry, if <i>rec.pending</i> is <i>REC_PENDING_HOST_CALL</i> , then GPR values X0 to X30 are copied from <i>rec_enter.gprs[0..30]</i> to the <i>RsiHostCall</i> data structure.
R _{YWHKC}	On REC entry, if writing to the <i>RsiHostCall</i> data structure fails due to the target IPA not being mapped then a REC exit to Data Abort results.
R _{TZVNK}	On REC entry, if writing to the <i>RsiHostCall</i> data structure succeeds then <i>rec.pending</i> is <i>REC_PENDING_NONE</i> .

R_{NLVXB} On REC entry, if RMM access to `rec_enter` causes a GPF then the `RMI_REC_ENTER` command fails with `RMI_ERROR_INPUT`.

See also:

- [A4.3.3 General purpose registers saved on REC exit](#)
- [A4.3.4.3 REC exit due to Data Abort](#)
- [A4.3.7 REC exit due to PSCI](#)
- [A4.3.9 REC exit due to Host call](#)
- [A4.5 Host call](#)

A4.2.3 REC entry following REC exit due to Data Abort

R_{TWMDB} On REC entry, if `rec_enter.flags.inject_sea == RMI_INJECT_SEA` then the value of `rec_enter.flags.emul_mmio` is ignored.

R_{BWZKH} On REC entry, if the most recent exit from the target REC was a REC exit due to Emulatable Data Abort and `rec_enter.flags.emul_mmio == RMI_EMULATED_MMIO`, then the return address is the next instruction following the faulting instruction.

R_{SCJWG} On REC entry, if the most recent exit from the target REC was a REC exit due to Emulatable Data Abort and the Realm memory access was a read and `rec_enter.flags.emul_mmio == RMI_EMULATED_MMIO`, then the register indicated by `ESR_EL2.ISS.SRT` is set to `rec_enter.gprs[0]`.

I_{KNFDT} On execution of `RMI_REC_ENTER`, if the most recent exit from the target REC was not a REC exit due to Emulatable Data Abort and `rec_enter.flags.emul_mmio == RMI_EMULATED_MMIO`, then the `RMI_REC_ENTER` command fails.

R_{LJWRK} On REC entry, if the most recent exit from the target REC was a REC exit due to Data Abort at an Unprotected IPA and `rec_enter.flags.inject_sea == RMI_INJECT_SEA`, then a Synchronous External Abort is taken to the Realm.

See also:

- [A4.3.4.3 REC exit due to Data Abort](#)
- [A4.4 Emulated Data Aborts](#)
- [A5.2.7 Realm access to an Unprotected IPA](#)
- [A5.2.8 Synchronous External Aborts](#)

A4.3 REC exit

This section describes REC exit.

See also:

- [A4.2 REC entry](#)
- [B4.3.30 RMI_REC_ENTER command](#)

A4.3.1 RmiRecExit object

D _{PBDCB}	An <i>RmiRecExit</i> object is a data structure used to pass values from the RMM to the Host on REC exit.
I _{VHJTL}	An <i>RmiRecExit</i> object is stored in the <i>RecRun</i> object which is passed by the Host as an input to the <i>RMI_REC_ENTER</i> command.
I _{JKWPB}	On REC exit, execution state is saved from the PE to the REC object and to the <i>RmiRecExit</i> object.
I _{ZSCNM}	An <i>RmiRecExit</i> object contains attributes which are used to manage Realm virtual interrupts and Realm timers.
D _{FFCMN}	The attributes of an <i>RmiRecExit</i> object are summarized in the following table.

Name	Byte offset	Type	Description
exit_reason	0x0	RmiRecExitReason	Exit reason
esr	0x100	Bits64	Exception Syndrome Register
far	0x108	Bits64	Fault Address Register
hpfar	0x110	Bits64	Hypervisor IPA Fault Address register
rtt_tree	0x118	UInt64	Index of RTT tree active at time of the exit
rtt_level	0x120	Int64	Level of requested RTT
gprs[31]	0x200	Bits64	Registers
gicv3_hcr	0x300	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0x308	Bits64	GICv3 List Register values
gicv3_misr	0x388	Bits64	GICv3 Maintenance Interrupt State Register value
gicv3_vmcr	0x390	Bits64	GICv3 Virtual Machine Control Register value
cntp_ctl	0x400	Bits64	Counter-timer Physical Timer Control Register value
cntp_cval	0x408	Bits64	Counter-timer Physical Timer CompareValue Register value
cntv_ctl	0x410	Bits64	Counter-timer Virtual Timer Control Register value
cntv_cval	0x418	Bits64	Counter-timer Virtual Timer CompareValue Register value
ripas_base	0x500	Bits64	Base IPA of target region for pending RIPAS change
ripas_top	0x508	Bits64	Top IPA of target region for pending RIPAS change

Name	Byte offset	Type	Description
ripas_value	0x510	RmiRipas	RIPAS value of pending RIPAS change
s2ap_base	0x520	Bits64	Base IPA of target region for pending S2AP change
s2ap_top	0x528	Bits64	Top IPA of target region for pending S2AP change
vdev_id	0x530	Bits64	Virtual device ID
imm	0x600	Bits16	Host call immediate value
plane	0x608	UInt64	Plane index
vdev	0x610	Address	VDEV which triggered REC exit due to VDEV communication
vdev_action	0x618	RmiVdevAction	Action which triggered REC exit due to VDEV communication
dev_mem_base	0x620	Bits64	Base IPA of target region for device memory mapping validation
dev_mem_top	0x628	Bits64	Top IPA of target region for device memory mapping validation
dev_mem_pa	0x630	Address	Base PA of device memory region
pmu_ovf_status	0x700	RmiPmuOverflowStatus	PMU overflow status

I_{FQZXZ} In this chapter, both `rec_exit` and “the `RmiRecExit` object” refer to the `RmiRecExit` object which is provided to the `RMI_REC_ENTER` command.

R_{PNWZV} On REC exit, all `rec_exit` fields are zero unless specified otherwise.

See also:

- [A2.3 Realm Execution Context](#)
- [A4.2.1 RmiRecEnter object](#)
- [A4.5 Host call](#)
- [Chapter A6 Realm interrupts and timers](#)
- [Chapter A8 Realm debug and performance monitoring](#)
- [B4.4.33 RmiRecExit type](#)

A4.3.2 Realm exit reason

I_{DYWHJ} On return from the `RMI_REC_ENTER` command, the reason for the REC exit is indicated by `rec_exit.exit_reason` and `rec_exit.esr`.

See also:

- [B4.4.34 RmiRecExitReason type](#)

A4.3.3 General purpose registers saved on REC exit

R_{PBKVB} On REC exit due to PSCI, all of the following are true:

- `rec_exit.gprs[0]` contains the PSCI FID.
- `rec_exit.gprs[1..3]` contain the corresponding PSCI arguments. If the PSCI command has fewer than 3 arguments, the remaining values contain zero.

- GPR values X7 to X30 are saved from the PE to the REC object.

R_{FNZKM} On REC exit for any reason which is not REC exit due to PSCI, GPR values X0 to X30 are saved from the PE to the REC.

R_{MZGPT} On REC exit for any reason which is neither REC exit due to Host call nor REC exit due to PSCI, `rec_exit.gprs` is zero.

R_{FRGVT} On REC exit, if RMM access to `rec_exit` causes a GPF then the RMI_REC_ENTER command fails with RMI_ERROR_INPUT.

See also:

- [A4.2.2 General purpose registers restored on REC entry](#)
- [A4.3.7 REC exit due to PSCI](#)
- [A4.3.9 REC exit due to Host call](#)

A4.3.4 REC exit due to synchronous exception

I_{SNDHF} A synchronous exception taken to R-EL2 can cause a REC exit.

I_{RPSNC} The following table summarises the behavior of synchronous exceptions taken to R-EL2.

Exception class	Behavior
Trapped WFI or WFE instruction execution	REC exit due to WFI or WFE
HVC instruction execution in AArch64 state	Unknown exception taken to Realm
SMC instruction execution in AArch64 state	One of: <ul style="list-style-type: none"> • REC exit due to PSCI • RSI command handled by RMM, followed by return to Realm
Trapped MSR, MRS or System instruction execution in AArch64 state	Emulated by RMM, followed by return to Realm
Instruction Abort from a lower Exception level	REC exit due to Instruction Abort
Data Abort from a lower Exception level	REC exit due to Data Abort

R_{YLFMD} Realm execution of an SMC which is not part of one of the following ABIs results in a return value of SMCCC_NOT_SUPPORTED:

- PSCI
- RSI

See also:

- [A4.5 Host call](#)
- [Chapter B5 Realm Services Interface](#)
- [Chapter B6 Power State Control Interface](#)

A4.3.4.1 REC exit due to WFI or WFE

D_{GLHPX} A REC exit due to WFI or WFE is a REC exit due to WFI, WFIT, WFE or WFET instruction execution in a Realm.

R_{VTJQF} On WFI or WFIT instruction execution in a Realm, a REC exit due to WFI or WFE is caused if `rec_enter.trap_wfi` is RMI_TRAP.

R_{GBNGW} On WFE or WFET instruction execution in a Realm, a REC exit due to WFI or WFE is caused if `rec_enter.trap_wfe` is RMI_TRAP.

R_{YQWST}

On REC exit due to WFI or WFE, all of the following are true:

- `rec_exit.exit_reason` is `RMI_EXIT_SYNC`.
- `rec_exit.esr.EC` contains the value of `ESR_EL2.EC` at the time of the Realm exit.
- `rec_exit.esr.ISS.TI` contains the value of `ESR_EL2.ISS.TI` at the time of the Realm exit.
- All other `rec_exit` fields except for `rec_exit.givc3_*`, `rec_exit_cnt*` and `rec_exit.pmu_ovf_status` are zero.

R_{BPYBC}

On REC exit due to WFI or WFE, if the exit was caused by WFET or WFIT instruction execution then `rec_exit.gpr[0]` contains the timeout value.

See also:

- [A6.1 Realm interrupts](#)
- [A6.2 Realm timers](#)
- [A8.1 Realm PMU](#)

A4.3.4.2 REC exit due to Instruction Abort

D_{GYQXK}

A *REC exit due to Instruction Abort* is a REC exit due to a Realm instruction fetch from a Protected IPA for which either of the following is true:

- HIPAS is UNASSIGNED and RIPAS is RAM
- RIPAS is DESTROYED

R_{MGWRC}

On REC exit due to Instruction Abort, all of the following are true:

- `rec_exit.exit_reason` is `RMI_EXIT_SYNC`.
- `rec_exit.esr.EC` contains the value of `ESR_EL2.EC` at the time of the Realm exit.
- `rec_exit.esr.ISS.SET` contains the value of `ESR_EL2.ISS.SET` at the time of the Realm exit.
- `rec_exit.esr.ISS.EA` contains the value of `ESR_EL2.ISS.EA` at the time of the Realm exit.
- `rec_exit.esr.ISS.IFSC` contains the value of `ESR_EL2.ISS.IFSC` at the time of the Realm exit.
- `rec_exit.hpfar` contains the value of `HPFAR_EL2` at the time of the Realm exit.
- `rec_exit.rtt_tree` contains the index of the RTT tree within which the contents of an RTTE cause the Realm to exit.
- All other `rec_exit` fields except for `rec_exit.givc3_*`, `rec_exit_cnt*` and `rec_exit.pmu_ovf_status` are zero.

I_{HMFEM}

`HPFAR_EL2.FIPA` does not include the lowest 12 bits of the faulting IPA. `rec_exit.hpfar` therefore only reveals the Realm's access patterns at a granularity of 4KB. If support was added to this specification for Granule sizes larger than 4KB, `rec_exit.hpfar` would need to be masked accordingly.

See also:

- [A5.2.2 Realm IPA state](#)
- [A5.2.3 Realm access to a Protected IPA](#)
- [A6.1 Realm interrupts](#)
- [A6.2 Realm timers](#)
- [A8.1 Realm PMU](#)

A4.3.4.3 REC exit due to Data Abort

D_{CYRMT}

A *REC exit due to Emulatable Data Abort* is a REC exit due to a Realm data access to one of the following:

- an Unprotected IPA whose HIPAS is UNASSIGNED_NS, where the access caused `ESR_EL2.ISS.ISV` to be set to '1'
- an Unprotected IPA whose HIPAS is ASSIGNED_NS, where the access caused a stage 2 permission fault and caused `ESR_EL2.ISS.ISV` to be set to '1'

D_{MTZMC}

A *REC exit due to Non-emulatable Data Abort* is a REC exit due to a Realm data access to one of the following:

- an Unprotected IPA whose HIPAS is UNASSIGNED_NS, where the access caused `ESR_EL2.ISS.ISV` to be set to '0'

- an Unprotected IPA whose HIPAS is ASSIGNED_NS, where the access caused a stage 2 permission fault and caused `ESR_EL2.ISS.ISV` to be set to '0'
- a Protected IPA whose HIPAS is UNASSIGNED and whose RIPAS is RAM
- a Protected IPA whose RIPAS is DESTROYED.

`R_RYVFL`

On REC exit due to Data Abort, all of the following are true:

- `rec_exit.exit_reason` is `RMI_EXIT_SYNC`.
- `rec_exit.esr.EC` contains the value of `ESR_EL2.EC` at the time of the Realm exit.
- `rec_exit.esr.ISS.SET` contains the value of `ESR_EL2.ISS.SET` at the time of the Realm exit.
- `rec_exit.esr.ISS.FnV` contains the value of `ESR_EL2.ISS.FnV` at the time of the Realm exit.
- `rec_exit.esr.ISS.EA` contains the value of `ESR_EL2.ISS.EA` at the time of the Realm exit.
- `rec_exit.esr.ISS.DFSC` contains the value of `ESR_EL2.ISS.DFSC` at the time of the Realm exit.
- `rec_exit.hpfar` contains the value of `HPFAR_EL2` at the time of the Realm exit.
- `rec_exit.rtt_tree` contains the index of the RTT tree within which the contents of an RTTE cause the Realm to exit.

On REC exit due to Emulatable Data Abort, all of the following are true:

- `rec.emulatable_abort` is `EMULATABLE_ABORT`.
- `rec_exit.esr.ISS.ISV` contains the value of `ESR_EL2.ISS.ISV` at the time of the Realm exit.
- `rec_exit.esr.ISS.SAS` contains the value of `ESR_EL2.ISS.SAS` at the time of the Realm exit.
- `rec_exit.esr.ISS.SF` contains the value of `ESR_EL2.ISS.SF` at the time of the Realm exit.
- `rec_exit.esr.ISS.WnR` contains the value of `ESR_EL2.ISS.WnR` at the time of the Realm exit.
- `rec_exit.far` contains the value of `FAR_EL2` at the time of the Realm exit, with bits more significant than the size of a Granule masked to zero.

On REC exit due to Non-emulatable Data Abort at an Unprotected IPA, all of the following are true:

- `rec_exit.esr.IL` contains the value of `ESR_EL2.IL` at the time of the Realm exit.

On REC exit due to Data Abort, all other `rec_exit` fields except for `rec_exit.givc3*`, `rec_exit.cnt*` and `rec_exit.pmu_ovf_status` are zero.

`X_XHXJC`

On REC exit due to Emulatable Data Abort, `ESR_EL2.ISS.SSE` is not propagated to the Host. This is because this field is used to emulate sign extension on loads, which must be performed by the RMM so that the Realm can rely on architecturally correct behavior of the virtual execution environment.

`X_HSWFR`

On REC exit due to Emulatable Data Abort, the Host can calculate the faulting IPA from the `rec_exit.hpfar` and `rec_exit.far` values.

`I_WCYNY`

`HPFAR_EL2.FIPA` does not include the lowest 12 bits of the faulting IPA. `rec_exit.hpfar` therefore only reveals the Realm's access patterns at a granularity of 4KB. If support was added to this specification for Granule sizes larger than 4KB, `rec_exit.hpfar` would need to be masked accordingly.

`R_FFNHW`

On REC exit due to Emulatable Data Abort, if the Realm memory access was a write, `rec_exit.gprs[0]` contains the value of the register indicated by `ESR_EL2.ISS.SRT` at the time of the Realm exit.

`R_QBTFR`

On REC exit not due to Emulatable Data Abort, `rec.emulatable_abort` is `NOT_EMULATABLE_ABORT`.

See also:

- [A4.2.3 REC entry following REC exit due to Data Abort](#)
- [A4.4 Emulated Data Aborts](#)
- [A5.2.1 Realm IPA space](#)
- [A5.2.3 Realm access to a Protected IPA](#)
- [A5.2.7 Realm access to an Unprotected IPA](#)
- [A6.1 Realm interrupts](#)
- [A6.2 Realm timers](#)
- [A8.1 Realm PMU](#)

A4.3.5 REC exit due to IRQ

D_{YLWXX} A *REC exit due to IRQ* is a REC exit due to an IRQ exception which should be handled by the Host.

R_{TYJSX} On REC exit due to IRQ, `rec_exit.exit_reason` is `RMI_EXIT_IRQ`.

R_{CSQXV} On REC exit due to IRQ, `rec_exit.esr` is zero.

See also:

- [Chapter A6 Realm interrupts and timers](#)

A4.3.6 REC exit due to FIQ

D_{ZTYMM} A *REC exit due to FIQ* is a REC exit due to an FIQ exception which should be handled by the Host.

R_{PDSBD} On REC exit due to FIQ, `rec_exit.exit_reason` is `RMI_EXIT_FIQ`.

R_{GXZRF} On REC exit due to FIQ, `rec_exit.esr` is zero.

See also:

- [Chapter A6 Realm interrupts and timers](#)

A4.3.7 REC exit due to PSCI

I_{ZSGFP} A PSCI function executed by a Realm is either:

- handled by the RMM, returning to the Realm, or
- forwarded by the RMM to the Host via a *REC exit due to PSCI*.

D_{RFTQD} A *REC exit due to PSCI* is a REC exit due to Realm PSCI function execution by SMC instruction which was forwarded by the RMM to the Host.

I_{VBJXY} The following table summarises the behavior of PSCI function execution by a Realm.

PSCI functions not listed in this table are not supported. Calling a non-supported PSCI function results in a return value of `PSCI_NOT_SUPPORTED`.

PSCI function	Can result in REC exit due to PSCI	Requires Host to call <code>RMI_PSCI_COMPLETE</code>
PSCI_VERSION	No	-
PSCI_FEATURES	No	-
PSCI_CPU_SUSPEND	Yes	No
PSCI_CPU_OFF	Yes	No
PSCI_CPU_ON	Yes	Yes
PSCI_AFFINITY_INFO	Yes	Yes
PSCI_SYSTEM_OFF	Yes	No
PSCI_SYSTEM_RESET	Yes	No

R_{NTZNJ} On REC exit due to PSCI, `rec_exit.exit_reason` is `RMI_EXIT_PSCI`.

R_{SXGJK} On REC exit due to PSCI, `rec_exit.gpr`s contains sanitised parameters from the PSCI call.

R_{YTDGT} On REC exit due to PSCI, if the command arguments include an MPIDR value, `rec.pending` is set to `REC_PENDING_PSCI`. Otherwise, `rec.pending` is set to `REC_PENDING_NONE`.

I_KKFMQ

Following REC exit due to PSCI, if `rec.pending` is `REC_PENDING_PSCI`, the Host must complete the request by calling the `RMI_PSCI_COMPLETE` command, prior to re-entering the REC.

In the call to `RMI_PSCI_COMPLETE`, the Host provides the target REC, which corresponds to the MPIDR value provided by the Realm. This is necessary because the RMM does not maintain a mapping from MPIDR values to REC addresses. The RMM validates that the REC provided by the Host matches the MPIDR value.

In the call to `RMI_PSCI_COMPLETE`, the Host provides a PSCI status value, which the RMM handles as follows:

- If the Host provides `PSCI_SUCCESS`, the RMM performs the PSCI operation requested by the Realm. The result of the PSCI operation is recorded in the REC and returned to the Realm on the next entry to the calling REC.
- If the Host provides a status value other than `PSCI_SUCCESS`, the RMM validates that the status code is permitted for the PSCI operation requested by the Realm. If the status code is permitted, it is recorded in the REC and returned to the Realm on the next entry to the calling REC.

See also:

- [A4.3.3 General purpose registers saved on REC exit](#)
- [B3.53 PsciReturnCodePermitted function](#)
- [B4.3.21 RMI_PSCI_COMPLETE command](#)
- [Chapter B6 Power State Control Interface](#)
- [D1.4 PSCI flows](#)

A4.3.8 REC exit due to RIPAS change pending

D_JGCVY

A REC exit due to RIPAS change pending is a REC exit due to the Realm issuing a RIPAS change request.

R_QSSKK

On REC exit due to RIPAS change pending, all of the following are true:

- `rec_exit.exit_reason` is `RMI_EXIT_RIPAS_CHANGE`.
- `rec_exit.ripas_base` is the base IPA of the region on which a RIPAS change is pending.
- `rec_exit.ripas_top` is the top IPA of the region on which a RIPAS change is pending.
- `rec_exit.ripas_value` is the requested RIPAS value.
- `rec.ripas_addr` is the base IPA of the region on which a RIPAS change is pending.
- `rec.ripas_top` is the top IPA of the region on which a RIPAS change is pending.
- `rec.ripas_value` is the requested RIPAS value.

I_MCKKH

On REC exit due to RIPAS change pending:

- `rec_exit` holds the base IPA and the size of the region on which a RIPAS change is pending. These values inform the Host of the bounds of the RIPAS change request.
- `rec` holds the next IPA to be processed in a RIPAS change, and the top of the requested RIPAS change region. These values are used by the RMM to enforce that the `RMI_RTT_SET_RIPAS` command can only apply RIPAS change within the bounds of the RIPAS change request, and to report the progress of the RIPAS change to the Realm on the next REC entry.

See also:

- [A2.3.2 REC attributes](#)
- [A5.4 RIPAS change](#)

A4.3.9 REC exit due to Host call

D_WFZXK

A REC exit due to Host call is a REC exit due to `RSI_HOST_CALL` execution in a Realm.

R_GTJRP

On REC exit due to Host call, all of the following are true:

- `rec.pending` is `REC_PENDING_HOST_CALL`.
- `rec_exit.exit_reason` is `RMI_EXIT_HOST_CALL`.

- `rec_exit.imm` contains the immediate value passed to the `RSI_HOST_CALL` command.
- `rec_exit.plane` contains the index of the Plane which executed the `RSI_HOST_CALL` command.
- `rec_exit.gprs[0..30]` contain the register values passed to the `RSI_HOST_CALL` command.
- All other `rec_exit` fields except for `rec_exit.givc3_*`, `rec_exit_cnt*` and `rec_exit.pmu_ovf_status` are zero.

See also:

- [A4.5 Host call](#)
- [A6.1 Realm interrupts](#)
- [A6.2 Realm timers](#)
- [A8.1 Realm PMU](#)
- [B5.3.4 RSI_HOST_CALL command](#)

A4.3.10 REC exit due to SError

D_{PGMHP}

A REC exit due to SError is a REC exit due to an SError interrupt during Realm execution.

R_{LRCFP}

On REC exit due to SError, all of the following occur:

- `rec_exit.exit_reason` is `RMI_EXIT_SERROR`.
- `rec_exit.esr.EC` contains the value of `ESR_EL2.EC` at the time of the Realm exit.
- `rec_exit.esr.ISS.IDS` contains the value of `ESR_EL2.ISS.IDS` at the time of the Realm exit.
- `rec_exit.esr.ISS.AET` contains the value of `ESR_EL2.ISS.AET` at the time of the Realm exit.
- `rec_exit.esr.ISS.EA` contains the value of `ESR_EL2.ISS.EA` at the time of the Realm exit.
- `rec_exit.esr.ISS.DFSC` contains the value of `ESR_EL2.ISS.DFSC` at the time of the Realm exit.
- All other `rec_exit` fields except for `rec_exit.givc3_*`, `rec_exit_cnt*` and `rec_exit.pmu_ovf_status` are zero.

See also:

- [A6.1 Realm interrupts](#)
- [A6.2 Realm timers](#)
- [A8.1 Realm PMU](#)

A4.3.11 REC exit due to S2AP change pending

D₀₀₃₄

A REC exit due to S2AP change pending is a REC exit due to the Realm issuing an S2AP change request.

R₀₀₃₅

On REC exit due to S2AP change pending, all of the following are true:

- `rec_exit.exit_reason` is `RMI_EXIT_S2AP_CHANGE`.
- `rec_exit.s2ap_base` is the base IPA of the region on which an S2AP change is pending.
- `rec_exit.s2ap_top` is the top IPA of the region on which an S2AP change is pending.
- `rec.s2ap_addr` is the base IPA of the region on which an S2AP change is pending.
- `rec.s2ap_top` is the top IPA of the region on which an S2AP change is pending.
- `rec.s2ap_value` is the requested S2AP value.

I₀₀₃₆

On REC exit due to RIPAS change pending:

- `rec_exit` holds the base IPA and the size of the region on which an S2AP change is pending. These values inform the Host of the bounds of the RIPAS change request.
- `rec` holds the next IPA to be processed in an S2AP change, and the top of the requested S2AP change region. These values are used by the RMM to enforce that the `RMI_RTT_SET_S2AP` command can only apply S2AP change within the bounds of the S2AP change request, and to report the progress of the S2AP change to the Realm on the next REC entry.

R₀₀₃₇

On REC exit not due to S2AP change pending, all of the following are true:

- `rec.s2ap_addr` is 0

- `rec.s2ap_top` is 0

See also:

- [A2.3.2 REC attributes](#)
- [A10.3.3.2 Stage 2 Access Permissions change within a multi-Plane Realm](#)

A4.3.12 REC exit due to VDEV request

- D₀₀₃₈** A REC exit due to VDEV request is a REC exit due to the RMM requiring the Host to provide the VDEV object which matches a specified virtual device ID.
- R₀₀₃₉** On REC exit due to VDEV request, `rec_exit.exit_reason` is `RMI_EXIT_VDEV_REQUEST`.
- R₀₀₄₀** On REC exit due to VDEV request, `rec_exit.vdev_id` contains the requested virtual device ID.
- R₀₀₄₁** On REC exit due to VDEV request, `rec.vdev_pending` is set to `REC_PENDING_VDEV_REQUEST`.
- I₀₀₄₂** Following REC exit due to VDEV request, the Host must complete the request by calling the `RMI_VDEV_COMPLETE` command, prior to re-entering the REC.

In the call to `RMI_VDEV_COMPLETE`, the Host provides the target VDEV, which corresponds to the virtual device ID value provided by the Realm. This is necessary because the RMM does not maintain a mapping from virtual device IDs to VDEV objects. The RMM validates that the VDEV provided by the Host matches the virtual device ID value.

See also:

- [A9.2.2 Mapping from virtual device ID to VDEV object](#)
- [B4.3.51 RMI_VDEV_COMPLETE command](#)

A4.3.13 REC exit due to VDEV communication

- D₀₀₄₃** A REC exit due to VDEV communication is a REC exit due to `RSI_RDEV_CONTINUE` execution in a Realm.
- R₀₀₄₄** On REC exit due to VDEV communication, `rec_exit.exit_reason` is `RMI_EXIT_VDEV_COMM`.
- R₀₀₄₅** On REC exit due to VDEV communication, `rec_exit.vdev` identifies the VDEV which triggered the REC exit.
- R₀₀₄₆** On REC exit due to VDEV communication, the state of the VDEV which triggered the REC exit becomes `VDEV_COMMUNICATING`.
- R₀₀₄₇** On REC exit due to VDEV communication, the communication status of the VDEV which triggered the REC exit becomes `DEV_COMM_PENDING`.

See also:

- [A9.2 Communication between RMM and a device](#)
- [A9.5 Realm management of an assigned device interface](#)
- [B5.3.15 RSI_RDEV_CONTINUE command](#)

A4.3.14 REC exit due to device memory mapping validation

- D₀₀₄₈** A REC exit due to device memory mapping validation is a REC exit due to the Realm issuing a *device memory mapping validation request*.
- R₀₀₄₉** On REC exit due to device memory mapping validation, all of the following are true:
- `rec_exit.exit_reason` is `RMI_EXIT_DEV_MEM_MAP`.
 - `rec_exit.dev_mem_base` is the base IPA of the region on which device memory mapping validation is pending.
 - `rec_exit.dev_mem_top` is the top IPA of the region on which device memory mapping validation is pending.
 - `rec.dev_mem_addr` is the base IPA of the region on which device memory mapping validation is pending.

- `rec.dev_mem_top` is the top IPA of the region on which device memory mapping validation is pending.
- `rec_exit.dev_mem_pa` is the base PA of the device memory region.
- `rec.dev_mem_pa` is the base PA of the device memory region.

I_0050

On REC exit due to device memory mapping validation:

- `rec_exit` holds the base IPA and the size of the region on which device memory mapping validation is pending. These values inform the Host of the bounds of the device memory mapping validation request.
- `rec_exit` holds the base PA of the device memory region. This value enables the Host to create device memory mappings (by calling `RMI_DEV_MEM_MAP`) on demand.
- `rec` holds the next IPA to be processed in a device memory mapping validation, and the top of the requested IPA region. These values are used by the RMM to enforce that the `RMI_RTT_DEV_MEM_VALIDATE` command can only apply device memory mapping validation within the bounds of the device memory mapping validation request, and to report the progress of the device memory mapping validation to the Realm on the next REC entry.

See also:

- [A2.3.2 REC attributes](#)
- [A5.5 Device memory mapping validation](#)
- [A9.5.3 Realm validation of device memory mappings](#)

DRAFT

A4.4 Emulated Data Aborts

I_{SVYDC}

On REC exit due to Emulatable Data Abort, sufficient information is provided to the Host to enable it to emulate the access, for example to emulate a virtual peripheral.

On taking the REC exit, the Host can either

- Establish a mapping in the RTT, in which case it would want the Realm to re-attempt the access. In this case, on the next REC entry the Host sets `enter.flags.emul_mmio = RMI_NOT_EMULATED_MMIO`, which indicates that instruction emulation was not performed. This causes the return address to be the faulting instruction.
- Emulate the access. For an emulated write, the data is provided in `exit.gprs[0]`. For an emulated read, the data is provided in `enter.gprs[0]`. In this case, on the next REC entry the Host sets `enter.flags.emul_mmio = RMI_EMULATED_MMIO`, which indicates that the instruction was emulated. This causes the return address to be the address of the instruction which generated the Data Abort plus 4 bytes.

See also:

- [A4.2.3 REC entry following REC exit due to Data Abort](#)
- [A4.3.4.3 REC exit due to Data Abort](#)
- [A5.2.1 Realm IPA space](#)

A4.5 Host call

This section describes the programming model for Realm communication with the Host.

D_{YDJWT}

A *Host call* is a call made by the Realm to the Host, by execution of the `RSI_HOST_CALL` command.

I_{XNFKZ}

A Host call can be used by a Realm to make a hypercall.

R_{DNBQF}

On Realm execution of HVC, an Unknown exception is taken to the Realm.

See also:

- [A4.2.2 General purpose registers restored on REC entry](#)
- [A4.3.9 REC exit due to Host call](#)
- [B5.3.4 RSI_HOST_CALL command](#)
- [D1.3.2 Host call flow](#)

Chapter A5

Realm memory management

This section describes how Realm memory is managed. This includes:

- How the translation tables which describe the Realm's address space are managed by the Host.
- Properties of the Realm's address space, and of the memory which can be mapped into it.
- How faults caused by Realm memory accesses are handled.

See also:

- [A2.1.2 Realm execution environment](#)
- [D1.5 Realm memory management flows](#)
- [Chapter D2 Realm shared memory protocol](#)

A5.1 Realm memory management overview

Realm memory management can be viewed from one of two standpoints: the Realm and the Host.

From the Realm's point of view, the RMM provides security guarantees regarding the IPA space of the Realm and the memory which is mapped into it. These security guarantees are upheld via RSI commands which the Realm can execute in order to query the initial configuration and contents of its address space, and to modify properties of the address space at runtime.

From the Host's point of view, Realm memory management involves manipulating the stage 2 translation tables which describe the Realm's address space, and handling faults which are caused by Realm memory accesses. These operations are similar to those involved in managing the memory of a normal VM, but in the case of a Realm they are performed via execution of RMI commands.

See also:

- [A5.2 Realm view of memory management](#)
- [A5.3 Host view of memory management](#)

A5.2 Realm view of memory management

This section describes memory management from the Realm's point of view.

A5.2.1 Realm IPA space

I_{DLRZF}

The IPA space of a Realm is divided into two halves: Protected IPA space and Unprotected IPA space.

S_{LZHXC}

Software in a Realm should treat the most significant bit of an IPA as a protection attribute.

D_{KXGDV}

A *Protected IPA* is an address in the lower half of a Realm's IPA space. The most significant bit of a Protected IPA is 0.

D_{MRWGM}

An *Unprotected IPA* is an address in the upper half of a Realm's IPA space. The most significant bit of an Unprotected IPA is 1.

See also:

- [A2.1.3 Realm attributes](#)
- [A3.3 Realm LPA2 and IPA width](#)

A5.2.2 Realm IPA state

D_{WWCBD}

A Protected IPA has an associated *Realm IPA state* (RIPAS).

The RIPAS values are shown in the following table.

Name	Description
DESTROYED	Address which is inaccessible to the Realm due to an action taken by the Host.
DEV	Address where memory of an assigned Realm device is mapped.
EMPTY	Address where no Realm resources are mapped.
RAM	Address where private code or data owned by the Realm is mapped.

I_{VZCZV}

RIPAS values are stored in an RTT.

I_ZPNZT The Realm can query the RIPAS of an IPA range by executing RSI_IPA_STATE_GET.

See also:

- [A5.6 Realm Translation Table](#)
- [Chapter A9 Realm device assignment](#)
- [B5.3.5 RSI_IPA_STATE_GET command](#)

A5.2.3 Realm access to a Protected IPA

R_JVQQR Realm data access to a Protected IPA whose RIPAS is EMPTY causes a Synchronous External Abort taken to the Realm.

R_MKLSQD Realm instruction fetch from a Protected IPA whose RIPAS is EMPTY causes a Synchronous External Abort taken to the Realm.

R_QSQLF Realm data access to a Protected IPA whose RIPAS is RAM does not cause a Synchronous External Abort taken to the Realm.

I_PGHBQ Realm data access to a Protected IPA whose RIPAS is RAM can cause an REC exit due to Data Abort.

R_FCJCP Realm instruction fetch from a Protected IPA whose RIPAS is RAM does not cause a Synchronous External Abort taken to the Realm.

I_XHKQY Realm instruction fetch from a Protected IPA whose RIPAS is RAM can cause a REC exit due to Instruction Abort.

R_CLVKF Realm data access to a Protected IPA whose RIPAS is DESTROYED causes a REC exit due to Data Abort.

R_MZYQT Realm instruction fetch from a Protected IPA whose RIPAS is DESTROYED causes a REC exit due to Instruction Abort.

R_0051 Realm data access to a Protected IPA whose RIPAS is DEV does not cause a Synchronous External Abort taken to the Realm.

I_0052 Realm data access to a Protected IPA whose RIPAS is DEV can cause an REC exit due to Data Abort.

R_0053 Realm instruction fetch from a Protected IPA whose RIPAS is DEV causes a Synchronous External Abort taken to the Realm.

See also:

- [A4.3.4.2 REC exit due to Instruction Abort](#)
- [A4.3.4.3 REC exit due to Data Abort](#)
- [A5.2.4 RSI command access to a Protected IPA](#)
- [A5.2.8 Synchronous External Aborts](#)
- [A9.7 Device access to a Protected IPA](#)

A5.2.4 RSI command access to a Protected IPA

R_0054 Access by an RSI command to a Protected IPA whose RIPAS is EMPTY causes the command to fail and return an error to the Realm.

R_0055 Access by an RSI command to a Protected IPA whose RIPAS is not EMPTY is treated as a Realm data access to the same address, for the purposes of deciding whether the access causes a REC exit due to Data Abort.

See also:

- [A5.2.3 Realm access to a Protected IPA](#)

A5.2.5 Changes to RIPAS while Realm state is REALM_NEW

This section describes how the RIPAS of a Protected IPA can change while the Realm state is REALM_NEW.

I _{BSBHN}	For a Realm in the REALM_NEW state, the RIPAS of a Protected IPA can change to RAM due to Host execution of RMI_DATA_CREATE or RMI_RTT_INIT_RIPAS.
I _{BSGSW}	For a Realm in the REALM_NEW state, changing the RIPAS of a Protected IPA to RAM causes the RIM to be updated.
I _{YCPNY}	For a Realm in the REALM_NEW state, the RIPAS of a Protected IPA can change to DESTROYED due to Host execution of RMI_DATA_DESTROY or RMI_RTT_DESTROY.
I _{YXLCF}	For a Realm in the REALM_NEW state, changing the RIPAS of a Protected IPA to DESTROYED does not cause the RIM to be updated.

See also:

- [A5.4 RIPAS change](#)
- [A7.1.1 Realm Initial Measurement](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)
- [B4.3.42 RMI_RTT_INIT_RIPAS command](#)

A5.2.6 Changes to RIPAS while Realm state is REALM_ACTIVE

This section describes how the RIPAS of a Protected IPA can change while the Realm state is REALM_ACTIVE.

I _{NZXPB}	A Realm in the REALM_ACTIVE state can request the RIPAS of a region of Protected IPA space to be changed to EMPTY, RAM or DEV.
I _{RHXPF}	A Realm in the REALM_ACTIVE state cannot request the RIPAS of a region of Protected IPA space to be changed to DESTROYED.
I _{FRJJH}	For a Realm in the REALM_ACTIVE state, the RIPAS of a Protected IPA can change to EMPTY only in response to Realm execution of RSI_IPA_STATE_SET.
X _{HQLVY}	The fact that the Host cannot change the RIPAS of a Protected IPA to EMPTY without the Realm having consented to this change prevents the Host from injecting an SEA at a Protected IPA which has been configured to have a RIPAS of RAM, which could potentially trigger unexpected behavior in the Realm.
I _{HNFYR}	For a Realm in the REALM_ACTIVE state, the RIPAS of a Protected IPA can change to RAM only in response to Realm execution of RSI_IPA_STATE_SET.
I _{VVFMX}	On execution of RSI_IPA_STATE_SET, a Realm can optionally specify that the RIPAS change should only succeed if the current RIPAS is not DESTROYED.
X _{VXHBV}	An expected pattern for Realm creation is as follows:

1. Host populates an “initial image” range of Realm IPA space with measured content:

Host executes RMI_DATA_CREATE, establishing a mapping to physical memory, changing RIPAS to RAM and updating the RIM.

2. Host informs the Realm of the range of IPA space which should be considered by the Realm as DRAM. This is a superset of the IPA range populated in step 1. For unpopulated parts of this IPA range, the RIPAS is EMPTY.
3. Realm executes RSI_IPA_STATE_SET(ripas=RAM) for the DRAM IPA range described to it in step 2. Following this command, the desired state is:
 - a. For the initial image IPA range, the contents match those described by the RIM.
 - b. For the entire DRAM IPA range, RIPAS is RAM.

If at step 2, the Host were to execute RMI_DATA_DESTROY on a page within the initial image IPA range, its RIPAS would change to DESTROYED. The Host could then execute RMI_DATA_CREATE_UNKNOWN, with the result that contents of the initial image IPA range no longer match those described by the RIM.

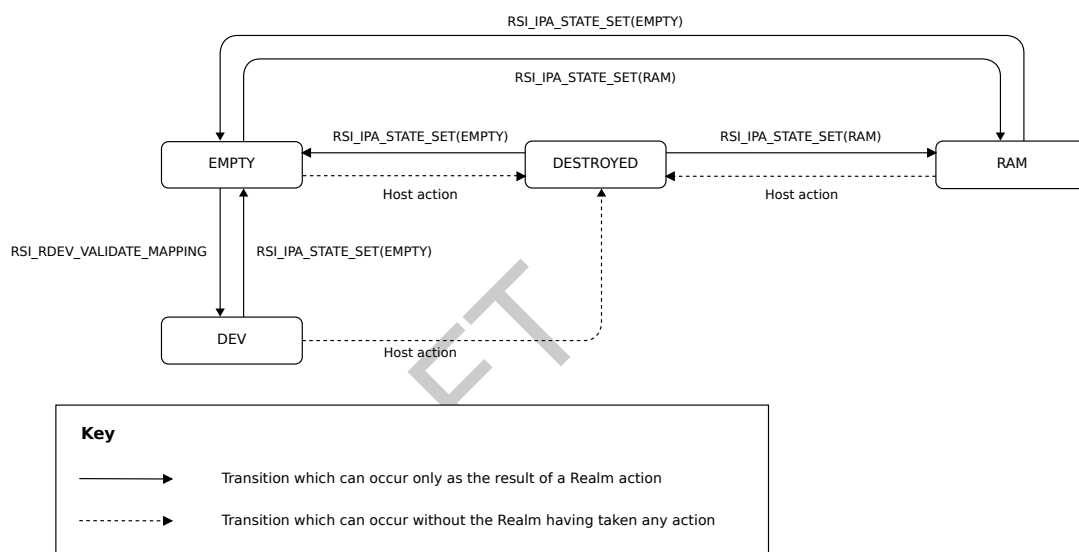
By specifying at step 3 that the RIPAS change should only succeed if the current RIPAS is not DESTROYED, the Realm is able to prevent loss of integrity within the initial image IPA range.

I₀₀₅₆ For a Realm in the REALM_ACTIVE state, the RIPAS of a Protected IPA can change to DEV only in response to Realm execution of RSI_RDEV_VALIDATE_MAPPING.

I_{KZVDC} For a Realm in the REALM_ACTIVE state, the RIPAS of a Protected IPA can change to DESTROYED due to Host execution of RMI_DATA_DESTROY or RMI_RTT_DESTROY.

X_{JJPHJ} The result of changing the RIPAS of a Protected IPA to DESTROYED is that subsequent Realm accesses to that address do not make forward progress. This is consistent with the principle that the RMM does not provide an availability guarantee to a Realm.

I_{NMMSG} The following diagram summarizes RIPAS changes which can occur when the Realm state is REALM_ACTIVE.



See also:

- [A5.4 RIPAS change](#)
- [A5.5 Device memory mapping validation](#)
- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)
- [B4.3.40 RMI_RTT_DEV_MEM_VALIDATE command](#)
- [B4.3.42 RMI_RTT_INIT_RIPAS command](#)
- [B5.3.6 RSI_IPA_STATE_SET command](#)
- [B5.3.24 RSI_RDEV_VALIDATE_MAPPING command](#)

A5.2.7 Realm access to an Unprotected IPA

I_{KQJML} An access by a Realm to an Unprotected IPA can result in a *Granule Protection Fault* (GPF).

The RMM does not ensure that the GPT entry of a Granule mapped at an Unprotected IPA permits access via Non-secure PAS.

S_{ZZBQF} Realm software must be able to handle taking a GPF during access to an Unprotected IPA.

I_{WCVBZ} Realm data access to an Unprotected IPA can cause a REC exit due to Data Abort.

I_{RNDTJ} On taking a REC exit due to Data Abort at an Unprotected IPA, the Host can inject a Synchronous External Abort to the Realm.

X _{MGBDH}	The Host can inject an SEA in response to an unexpected Realm data access to an Unprotected IPA.
I _{FVYCM}	Realm data access to an Unprotected IPA which caused <code>ESR_EL2.ISS.ISV</code> to be set to '1' can be emulated by the Host.
R _{XLSKP}	Realm instruction fetch from an Unprotected IPA causes a Synchronous External Abort taken to the Realm. See also: <ul style="list-style-type: none">• A4.2.3 REC entry following REC exit due to Data Abort• A4.3.4.3 REC exit due to Data Abort• A4.4 Emulated Data Aborts• A5.2.8 Synchronous External Aborts

A5.2.8 Synchronous External Aborts

R _{VKNJW}	When a Synchronous External Abort is taken to a Realm, <code>ESR_EL1.EA == '1'</code> .
--------------------	---

A5.2.9 Realm access outside IPA space

R _{GYVZQ}	If stage 1 translation is enabled, Realm access to an IPA which is greater than the IPA space of the Realm causes a stage 1 Address Size Fault taken to the Realm, with the fault status code indicating the level at which the fault occurred.
R _{LSJJR}	If stage 1 translation is disabled, Realm access to an IPA which is greater than the IPA space of the Realm causes a stage 1 level 0 Address Size Fault taken to the Realm.

A5.2.10 Summary of Realm IPA space properties

TPGKW

The following table summarizes the properties of Realm IPA space.

Realm IPA	Data access causes abort to Realm?	Data access causes REC exit due to Data Abort?	Instruction fetch causes abort to Realm?	Instruction fetch causes REC exit due to Instruction Abort?
Protected, RIPAS=EMPTY	Always (SEA)	Never	Always (SEA)	Never
Protected, RIPAS=RAM	Never	When HIPAS=UNASSIGNED	Never	When HIPAS=UNASSIGNED
Protected, RIPAS=DEV	Never	When HIPAS=UNASSIGNED	Always (SEA)	Never
Protected, RIPAS=DESTROYED	Never	Always	Never	Always
Unprotected	Host can inject SEA following REC exit due to Data Abort	When HIPAS=UNASSIGNED_NS or HIPAS=ASSIGNED_NS and access caused a stage 2 permission fault	Always (SEA)	Never
Outside Realm IPA space	Always (Address Size Fault)	Never	Always (Address Size Fault)	Never

See also:

- [A4.2.3 REC entry following REC exit due to Data Abort](#)

A5.2.11 Cache maintenance operations

RTZQDY

A data cache invalidate by set / way instruction executed by a Realm either has no effect, or performs a data cache clean and invalidate.

XXZRDW

This is to ensure that a Realm cannot invalidate a cache line owned by another Realm.

UVQMTB

Arm expects that the RMM will set HCR_EL2.VM == '1', which causes a data cache invalidate instruction executed at EL1 to perform a data cache clean and invalidate.

A5.3 Host view of memory management

This section describes memory management from the Host's point of view.

A5.3.1 Host IPA state

D_{YZTZJ} A Realm IPA has an associated *Host IPA state* (HIPAS).

The HIPAS values are shown in the following table.

Name	Description
HIPAS_ASSIGNED	Protected IPA which is associated with a DATA Granule.
HIPAS_ASSIGNED_DEV	Protected IPA which is associated with a DEV_MAPPED Granule.
HIPAS_ASSIGNED_NS	Unprotected IPA which is associated with a physical Granule.
HIPAS_ASSIGNED_VSMMU	Protected IPA which is associated with a VSMMU Granule.
HIPAS_UNASSIGNED	Protected IPA which is not associated with any Granule.
HIPAS_UNASSIGNED_NS	Unprotected IPA which is not associated with any Granule.

I_{TRSKJ} HIPAS values are stored in a Realm Translation Table (RTT).

I_{GZMKQ} HIPAS transitions are caused by execution of RMI commands.

I_{NQCGS} A mapping at a Protected IPA is valid if the HIPAS is ASSIGNED and the RIPAS is RAM.

I₀₀₅₇ A mapping at a Protected IPA is valid if the HIPAS is ASSIGNED_DEV and the RIPAS is DEV.

I₀₀₅₈ The RMM emulates access to a Protected IPA if the HIPAS is ASSIGNED_SMMU and the RIPAS is DEV.

└_{YMNSR}

The following table summarizes, for each combination of RIPAS and HIPAS for a Protected IPA:

- the translation table entry attributes, and
- the behavior which results from Realm access to that IPA.

Each TTD.X column refers to the value of the corresponding “X” field in the architecturally-defined Stage 2 translation table descriptor which is written by the RMM.

RIPAS	HIPAS	TTD.ADDR	TTD.NS	TTD.VALID	Data access	Instruction fetch
EMPTY	UNASSIGNED			0	SEA to Realm	SEA to Realm
EMPTY	ASSIGNED	DATA		0	SEA to Realm	SEA to Realm
EMPTY	ASSIGNED_DEV	DEV		0	SEA to Realm	SEA to Realm
EMPTY	ASSIGNED_VSMMUDEV			0	SEA to Realm	SEA to Realm
RAM	UNASSIGNED			0	REC exit due to Data Abort	REC exit due to Instruction Abort
RAM	ASSIGNED	DATA	0	1	Data access	Instruction fetch
RAM	ASSIGNED_DEV	DEV		0	REC exit due to Data Abort	REC exit due to Instruction Abort
RAM	ASSIGNED_VSMMUDEV			0	REC exit due to Data Abort	REC exit due to Instruction Abort
DESTROYED	UNASSIGNED			0	REC exit due to Data Abort	REC exit due to Instruction Abort
DESTROYED	ASSIGNED	DATA		0	REC exit due to Data Abort	REC exit due to Instruction Abort
DESTROYED	ASSIGNED_DEV	DEV		0	REC exit due to Data Abort	REC exit due to Instruction Abort
DESTROYED	ASSIGNED_VSMMUDEV			0	REC exit due to Data Abort	REC exit due to Instruction Abort
DEV	UNASSIGNED			0	REC exit due to Data Abort	SEA to Realm
DEV	ASSIGNED	DATA		0	REC exit due to Data Abort	SEA to Realm
DEV	ASSIGNED_DEV	DEV		1	Device access	SEA to Realm
DEV	ASSIGNED_VSMMUDEV			0	Emulated by RMM	SEA to Realm

See also:

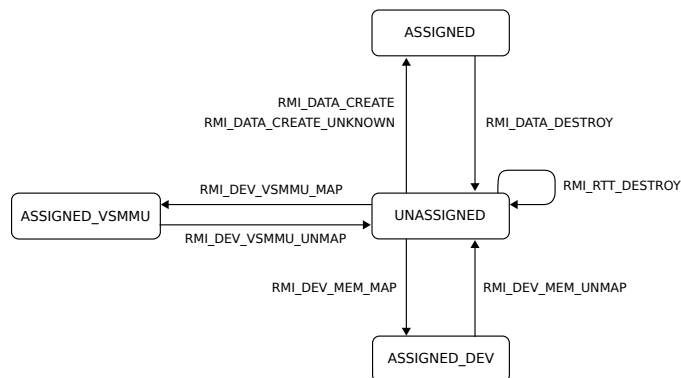
- [A5.2.3 Realm access to a Protected IPA](#)
- [A5.2.4 RSI command access to a Protected IPA](#)
- [A5.6 Realm Translation Table](#)
- [Chapter A9 Realm device assignment](#)
- [A9.6 Virtual SMMU](#)

A5.3.2 Changes to HIPAS while Realm state is REALM_NEW

This section describes how the HIPAS of a Protected IPA can change while the Realm state is REALM_NEW.

I_{YNFGD}

The following diagram summarizes HIPAS changes at a Protected IPA which can occur when the Realm state is REALM_NEW.



See also:

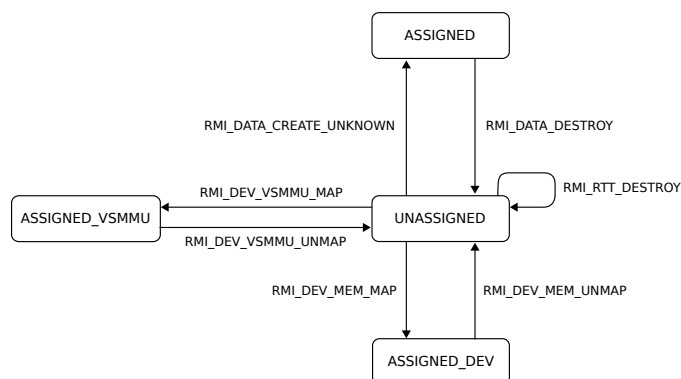
- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.4 RMI_DEV_MEM_MAP command](#)
- [B4.3.5 RMI_DEV_MEM_UNMAP command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)

A5.3.3 Changes to HIPAS while Realm state is REALM_ACTIVE

This section describes how the HIPAS of a Protected IPA can change while the Realm state is REALM_ACTIVE.

I_{WKZXY}

The following diagram summarizes HIPAS changes at a Protected IPA which can occur when the Realm state is REALM_ACTIVE.



See also:

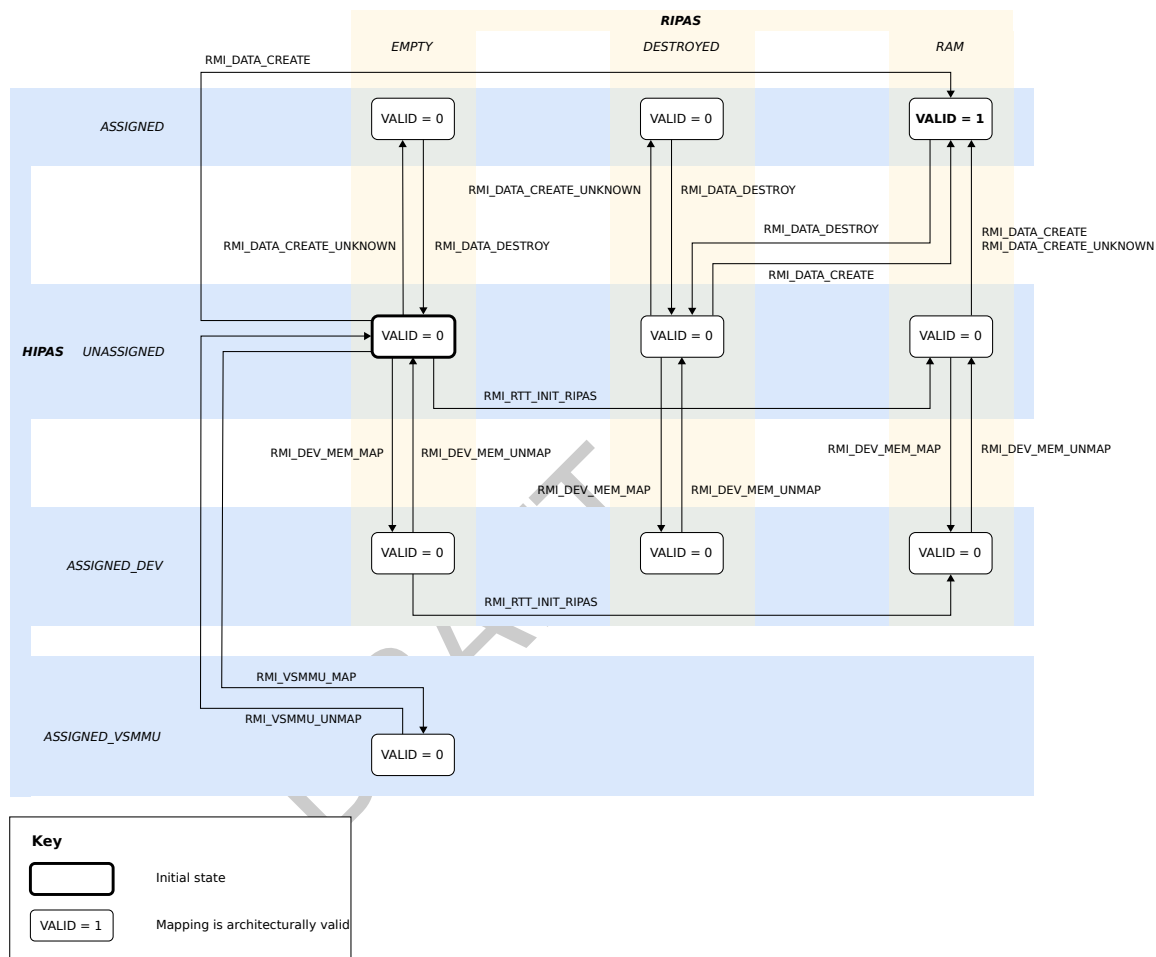
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.4 RMI_DEV_MEM_MAP command](#)
- [B4.3.5 RMI_DEV_MEM_UNMAP command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)

A5.3.4 Summary of changes to HIPAS and RIPAS of a Protected IPA

└─TJMCP

The following diagram summarizes HIPAS and RIPAS changes at a Protected IPA which can occur when the Realm state is NEW.

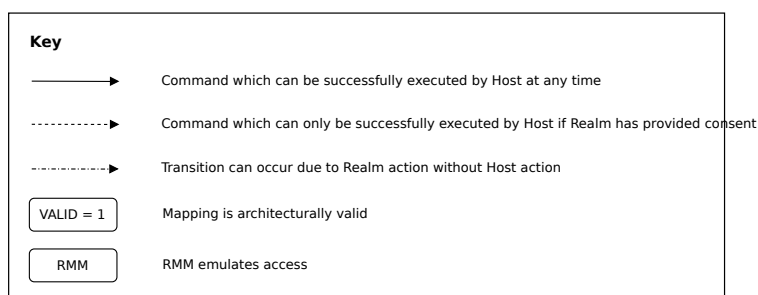
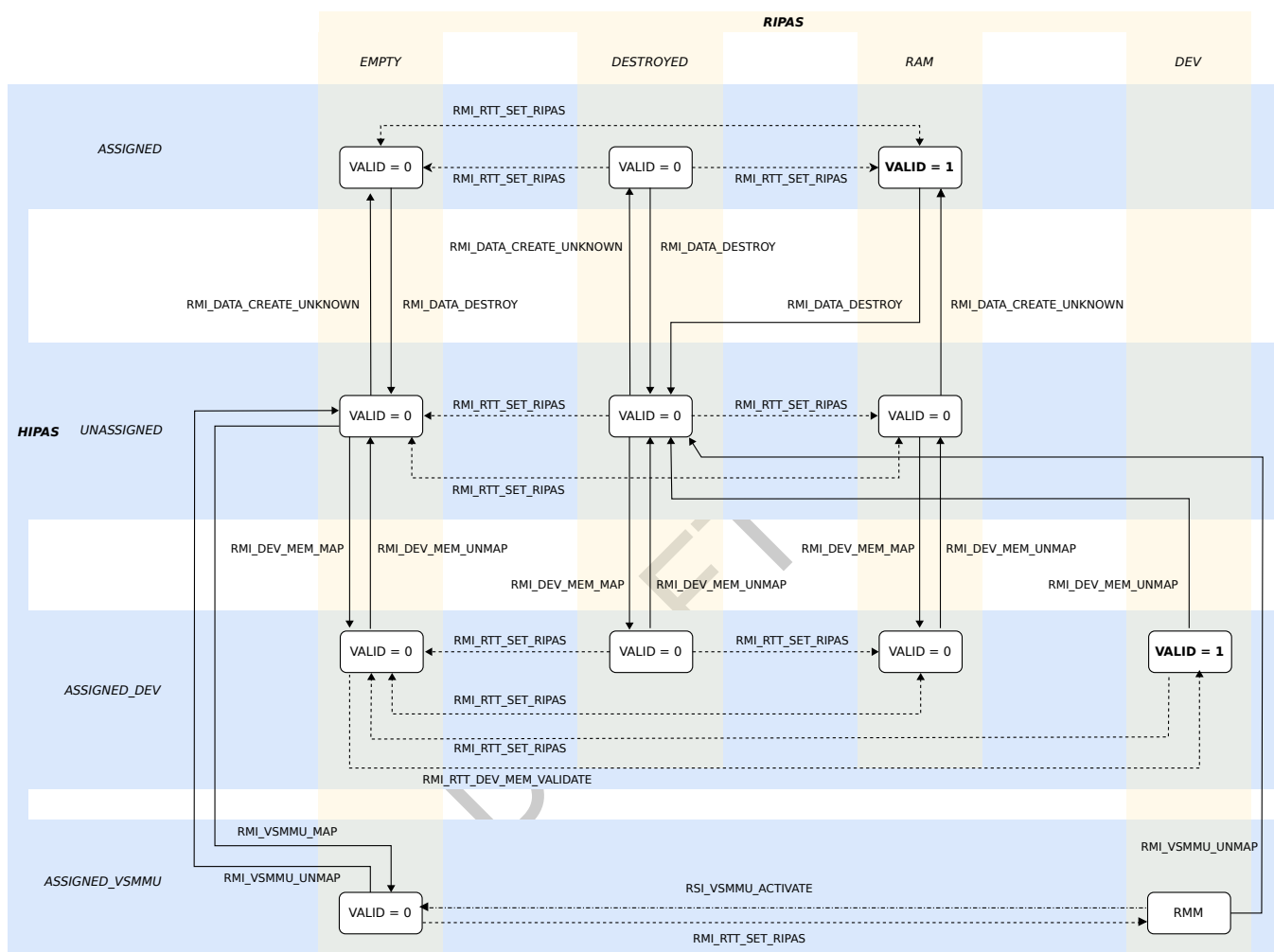
Transitions due to execution of RMI_RTT_DESTROY are omitted from the diagram. Execution of this command results in a transition to HIPAS=UNASSIGNED, RIPAS=DESTROYED.



I_VGKNJ

The following diagram summarizes HIPAS and RIPAS changes at a Protected IPA which can occur when the Realm state is REALM_ACTIVE.

Transitions due to execution of RMI_RTT_DESTROY are omitted from the diagram. Execution of this command results in a transition to HIPAS=UNASSIGNED, RIPAS=DESTROYED.



See also:

- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.4 RMI_DEV_MEM_MAP command](#)
- [B4.3.5 RMI_DEV_MEM_UNMAP command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)

- [B4.3.40 RMI_RTT_DEV_MEM_VALIDATE command](#)
- [B4.3.42 RMI_RTT_INIT_RIPAS command](#)
- [B4.3.45 RMI_RTT_SET_RIPAS command](#)

DRAFT

A5.3.5 Dependency of RMI command execution on RIPAS and HIPAS values

I_{HLHZS} The following table summarizes dependencies on RMI command execution on the current Protected IPA.

Command	Dependency on RIPAS	Dependency on HIPAS	New RIPAS	New HIPAS
RMI_DATA_CREATE	None	HIPAS is UNASSIGNED	RAM	ASSIGNED
RMI_DATA_CREATE_UNKNOWN	None	HIPAS is UNASSIGNED	Unchanged	ASSIGNED
RMI_DATA_DESTROY	If RIPAS is not RAM	HIPAS is ASSIGNED	Unchanged	UNASSIGNED
RMI_DATA_DESTROY	If RIPAS is RAM	HIPAS is ASSIGNED	DESTROYED	UNASSIGNED
RMI_RTT_CREATE	None	None	Unchanged	Unchanged
RMI_RTT_DESTROY	None	HIPAS of all entries is UNASSIGNED	DESTROYED	UNASSIGNED
RMI_RTT_DEV_MEM_VALIDATE	None	HIPAS of all entries is ASSIGNED_DEV	DEV	Unchanged
RMI_RTT_FOLD	RIPAS of all entries is identical	HIPAS of all entries is identical	Unchanged	Unchanged
RMI_RTT_INIT_RIPAS	None	HIPAS is UNASSIGNED	RAM	Unchanged
RMI_RTT_SET_RIPAS	Optionally, Realm may specify that RIPAS is not DESTROYED	None	As specified by Realm	Unchanged
RMI_DEV_MEM_MAP	None	HIPAS is UNASSIGNED	Unchanged	ASSIGNED_DEV
RMI_DEV_MEM_UNMAP	If RIPAS is not DEV	HIPAS is ASSIGNED_DEV	Unchanged	UNASSIGNED
RMI_DEV_MEM_UNMAP	If RIPAS is DEV	HIPAS is ASSIGNED_DEV	DESTROYED	UNASSIGNED
RMI_VSMMU_MAP	RIPAS is EMPTY	HIPAS is UNASSIGNED	Unchanged	ASSIGNED_VSMMU
RMI_VSMMU_UNMAP	If RIPAS is not DEV	HIPAS is ASSIGNED_VSMMU	Unchanged	UNASSIGNED
RMI_VSMMU_UNMAP	If RIPAS is DEV	HIPAS is ASSIGNED_VSMMU	DESTROYED	UNASSIGNED

$I_{WBR CN}$ Successful execution of RMI_DATA_CREATE_UNKNOWN does not depend on the RIPAS value of the target IPA.

I_{LCSVH} Successful execution of RMI_DATA_DESTROY does not depend on the RIPAS value of the target IPA.

I_{MMSBL} Successful execution of RMI_RTT_DESTROY does not depend on the RIPAS values of entries in the target RTT.

I_{TJCGT} Successful execution of RMI_RTT_FOLD does depend on the RIPAS values of entries in the target RTT.

I₀₀₅₉

Successful execution of RMI_DEV_MEM_UNMAP does not depend on the RIPAS value of the target IPA.

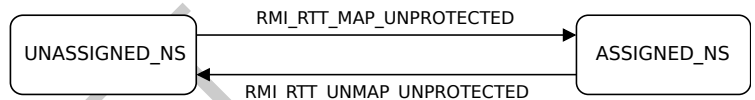
See also:

- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.4 RMI_DEV_MEM_MAP command](#)
- [B4.3.5 RMI_DEV_MEM_UNMAP command](#)
- [B4.3.38 RMI_RTT_CREATE command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)
- [B4.3.40 RMI_RTT_DEV_MEM_VALIDATE command](#)
- [B4.3.41 RMI_RTT_FOLD command](#)
- [B4.3.42 RMI_RTT_INIT_RIPAS command](#)
- [B4.3.45 RMI_RTT_SET_RIPAS command](#)
- [B4.3.59 RMI_VSMMU_MAP command](#)
- [B4.3.60 RMI_VSMMU_UNMAP command](#)

A5.3.6 Changes to HIPAS of an Unprotected IPA

I_{YNYBY}

The following diagram summarises HIPAS transitions for an Unprotected IPA.



See also:

- [A5.6 Realm Translation Table](#)
- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)
- [B4.3.42 RMI_RTT_INIT_RIPAS command](#)
- [B4.3.45 RMI_RTT_SET_RIPAS command](#)
- [B5.3.6 RSI_IPA_STATE_SET command](#)

A5.4 RIPAS change

D_{BTSQY} A *RIPAS change* is a process via which the RIPAS of a region of Protected IPA space is changed, for a Realm whose state is `REALM_ACTIVE`.

I_{KXXBV} A RIPAS change consists of actions taken first by the Realm, and then by the Host:

- The Realm issues a *RIPAS change request* by executing `RSI_IPA_STATE_SET`.
 - The input values to `RSI_IPA_STATE_SET` include:
 - * The requested IPA range: `[base, top)`
 - * The requested RIPAS value (either `EMPTY` or `RAM`)
 - * A flag which indicates whether a change from `DESTROYED` should be permitted
 - The RMM records these values in the REC, and then performs a REC exit due to RIPAS change pending.
- In response, the Host executes zero or more `RMI_RTT_SET_RIPAS` commands.
- If the requested RIPAS value was not `EMPTY` then at the next `RMI_REC_ENTER` the Host can optionally indicate that it rejects the RIPAS change request.

Output values from `RSI_IPA_STATE_SET` indicate:

- The top of the IPA range which has been modified by the command (`new_base`).
- If the requested RIPAS value was not `EMPTY`, whether the Host rejected the Realm request.

S_{CTTQV} Output values from `RSI_IPA_STATE_SET` are expected to be handled by the Realm as follows:

new_base	response	Meaning	Expected Realm action
<code>new_base == base</code>	<code>RSI_ACCEPT</code>	RIPAS change incomplete.	Call the command again, with <code>base = new_base</code> .
<code>base < new_base < top</code>	<code>RSI_ACCEPT</code>	RIPAS change incomplete.	Call the command again, with <code>base = new_base</code> .
<code>new_base == top</code>	<code>RSI_ACCEPT</code>	RIPAS change complete.	No further Realm action required.
<code>new_base == base</code>	<code>RSI_REJECT</code>	RIPAS change request rejected.	Depends on protocol agreed between Realm and Host, out of scope of this specification.
<code>base < new_base < top</code>	<code>RSI_REJECT</code>	RIPAS change to partial region <code>[base, new_base)</code> . Host rejected request to change RIPAS for region <code>[new_base, top)</code> .	Depends on protocol agreed between Realm and Host, out of scope of this specification.

I_{REFVTG} The RIPAS change process, together with the Realm Initial Measurement ensures that a Realm can always reliably determine the RIPAS of any Protected IPA.

I_{LPZWK} A RIPAS change is applied by one or more calls to the `RMI_RTT_SET_RIPAS` command.

I_{MMHMZ} Successful execution of `RMI_RTT_SET_RIPAS` targets an RTTE at address `rec.ripas_addr`.

I_{JHJGZ} On successful execution of `RMI_RTT_SET_RIPAS`, both of the following are set to the address of the next page whose RIPAS is to be modified:

- `rec.ripas_addr`
- The command output value

<code>I_{GXDDX}</code>	<p>If both of the following are true on successful execution of <code>RMI_RTT_SET_RIPAS</code></p> <ul style="list-style-type: none"> • The RIPAS change request indicated that a change from DESTROYED should not be permitted • A page <i>P</i> within the target IPA range has RIPAS value DESTROYED <p>then <code>rec.ripas_addr</code> and the command output value are both set to <i>P</i>.</p>
<code>I_{HXKPB}</code>	On REC entry following a REC exit due to RIPAS change, GPR values are updated to indicate for how much of the target IPA range the RIPAS change has been applied.
<code>S_{TZYZV}</code>	To complete a RIPAS change for a given target IPA range, a Realm should execute <code>RSI_IPA_STATE_SET</code> in a loop, until the value of <code>X1</code> reaches the top of the target IPA range.
<code>R_{LDMLC}</code>	On REC entry following a REC exit due to RIPAS change, <code>rec.ripas_response</code> is set to the value of <code>enter.flags.ripas_response</code> .
<code>I_{DRPPK}</code>	<p>If all of the following are true then the output value of <code>RSI_IPA_STATE_SET</code> indicates “Host rejected the request”:</p> <ul style="list-style-type: none"> • <code>rec.ripas_value</code> is RAM. • <code>rec.ripas_addr</code> is not equal to <code>rec.ripas_top</code>. • <code>rec.ripas_response</code> is REJECT. <p>Otherwise, the output value of <code>RSI_IPA_STATE_SET</code> indicates “Host accepted the request”.</p>
<code>S_{BZWWC}</code>	<p>Receipt of a rejection for a RIPAS change request whose parameters were valid is expected to be fatal for the Realm.</p> <p>See also:</p> <ul style="list-style-type: none"> • A2.3.2 REC attributes • A4.2 REC entry • A4.3.8 REC exit due to RIPAS change pending • A5.2.2 Realm IPA state • A7.1.1 Realm Initial Measurement • B3.71 RecRipasResponseToRsi function • B4.3.30 RMI_REC_ENTER command • B4.3.45 RMI_RTT_SET_RIPAS command • B5.3.6 RSI_IPA_STATE_SET command • D1.5.3 RIPAS change flow

A5.5 Device memory mapping validation

- D0060 A *Device memory mapping validation* is a process via which the addresses and attributes of mappings to device memory are checked for consistency with values expected by the Realm.
- I0061 The outcome of a successful Device memory mapping validation is a change of RIPAS to DEV.
- I0062 A Device memory mapping validation consists of actions taken first by the Realm, and then by the Host:
- The Realm issues a *Device memory mapping validation request* by executing RSI_RDEV_VALIDATE_MAPPING.
 - The input values to RSI_RDEV_VALIDATE_MAPPING include:
 - * The requested IPA range: [base, top)
 - * The base of the expected PA range
 - * Flags which indicate the expected memory attributes
 - The RMM records these values in the REC, and then performs a REC exit due to device memory mapping validation.
 - In response, the Host executes zero or more RMI_RTT_DEV_MEM_VALIDATE commands.
- Output values from RSI_RDEV_VALIDATE_MAPPING indicate:
- The top of the IPA range which has been modified by the command (new_base).
 - Whether the Host rejected the Realm request.
- S0063 Output values from RSI_RDEV_VALIDATE_MAPPING are expected to be handled by the Realm as follows:

new_base	response	Meaning	Expected Realm action
new_base == base	RSI_ACCEPT	Device memory mapping validation incomplete.	Call the command again, with base = new_base.
base < new_base < top	RSI_ACCEPT	Device memory mapping validation incomplete.	Call the command again, with base = new_base.
new_base == top	RSI_ACCEPT	Device memory mapping validation complete.	No further Realm action required.
new_base == base	RSI_REJECT	Device memory mapping validation request rejected.	Depends on protocol agreed between Realm and Host, out of scope of this specification.
base < new_base < top	RSI_REJECT	Device memory mapping validation to partial region [base, new_base). Host rejected request to validate device memory mapping for region [new_base, top).	Depends on protocol agreed between Realm and Host, out of scope of this specification.

- I0064 The Device memory mapping validation process, together with the Realm Initial Measurement ensures that a Realm can always reliably determine the RIPAS of any Protected IPA.
- I0065 A Device memory mapping validation is applied by one or more calls to the RMI_RTT_DEV_MEM_VALIDATE command.
- I0066 Successful execution of RMI_RTT_DEV_MEM_VALIDATE targets an RTTE at address `rec.dev_mem_addr`.
- I0067 On successful execution of RMI_RTT_DEV_MEM_VALIDATE, all of the following are set to the address of the next page whose device memory mapping is to be validated:
- `rec.dev_mem_addr`

- `rec.dev_mem_pa`
- The command output value

- I₀₀₆₈ On REC entry following a REC exit due to device memory mapping validation, GPR values are updated to indicate for how much of the target IPA range the device memory mapping validation request has been applied.
- S₀₀₆₉ To complete a Device memory mapping validation for a given target IPA range, a Realm should execute `RSI_RDEV_VALIDATE_MAPPING` in a loop, until the value of X1 reaches the top of the target IPA range.
- R₀₀₇₀ On REC entry following a REC exit due to device memory mapping validation, `rec.dev_mem_response` is set to the value of `enter.flags.dev_mem_response`.
- I₀₀₇₁ If all of the following are true then the output value of `RSI_RDEV_VALIDATE_MAPPING` indicates “Host rejected the request”:
- `rec.dev_mem_addr` is not equal to `rec.dev_mem_top`.
 - `rec.dev_mem_response` is `REJECT`.

Otherwise, the output value of `RSI_RDEV_VALIDATE_MAPPING` indicates “Host accepted the request”.

See also:

- [A2.3.2 REC attributes](#)
- [A4.2 REC entry](#)
- [A4.3.14 REC exit due to device memory mapping validation](#)
- [A5.2.2 Realm IPA state](#)
- [A7.1.1 Realm Initial Measurement](#)
- [Chapter A9 Realm device assignment](#)
- [B3.68 RecDevMemResponseToRsi function](#)
- [B4.3.30 RMI_REC_ENTER command](#)
- [B4.3.40 RMI_RTT_DEV_MEM_VALIDATE command](#)
- [B5.3.24 RSI_RDEV_VALIDATE_MAPPING command](#)
- [D1.5.3 RIPAS change flow](#)

A5.6 Realm Translation Table

This section introduces the stage 2 translation table used by a Realm.

A5.6.1 RTT overview

D _{FRNCX}	A <i>Realm Translation Table</i> (RTT) is an abstraction over an Armv8-A stage 2 translation table used by a Realm.
I _{MBCVZ}	The attributes and format of an Armv8-A stage 2 translation table are defined by the Armv8-A Virtual Memory System Architecture (VMSA) Arm Architecture Reference Manual for A-Profile architecture [3] .
R _{PXNHQ}	The translation granule size of an RTT is 4KB.
I _{TQVTP}	The RMM architecture can only be deployed on a hardware platform which implements a translation granule size of 4KB.
I _{PHGQQ}	The contents of an RTT are not directly accessible to the Host.
I _{FPLRL}	The contents of an RTT are manipulated using RMM commands. These commands allow the Host to manipulate the contents of the RTT used by a Realm, subject to constraints imposed by the RMM.
D _{QTZDW}	An <i>RTT entry</i> (RTTE) is an abstraction over an Armv8-A stage 2 translation table descriptor.
I _{VYLT}	An RTTE contains an output address which can point to one of the following: <ul style="list-style-type: none">• Another RTT• A DATA Granule which is owned by the Realm• Non-secure memory which is accessible to both the Realm and the Host

A5.6.2 RTT structure and configuration

D _{VHLWF}	An <i>RTT tree</i> is a hierarchical data structure composed of RTTs, connected via Table Descriptors.
I _{KNPNX}	An RTT contains an array of RTTEs.
D _{HYTCJ}	An <i>RTT level</i> is the depth of an RTT within an RTT tree.
I _{KKMSX}	An RTT does not have an intrinsic “level” attribute. The level of an RTT is determined by its position within an RTT tree.
D _{QSYBS}	The RTT level of the root of an RTT tree is called the <i>starting level</i> .
I _{SSDBT}	The maximum depth of an RTT tree depends on all of the following: <ul style="list-style-type: none">• whether LPA2 is selected when the Realm is created• the <code>rtt_level_start</code> attribute of the Realm• the <code>ipa_width</code> attribute of the Realm.

See also:

- [A2.1.3 Realm attributes](#)
- [A3.3 Realm LPA2 and IPA width](#)

A5.6.3 RTT starting level

I _{FDWZF}	The RTT starting level is set when a Realm is created.
I _{YCPMF}	The number of starting level RTTs is architecturally defined as a function of the Realm IPA width and the RTT starting level. See Arm Architecture Reference Manual for A-Profile architecture [3] for further details.
I _{RYNXB}	The address of the first starting level RTT is stored in the RTT base attribute of the owning Realm.

I_{XXWQW} The RTT base attribute is set when a Realm is created.

See also:

- [A2.1.3 Realm attributes](#)

A5.6.4 RTT entry

I_{ZBGGZ} An RTT entry (RTTE) is an abstraction over an Armv8-A stage 2 translation table descriptor. The attributes and format of an Armv8-A stage 2 translation table descriptor are defined by the Armv8-A Virtual Memory System Architecture (VMSA) [Arm Architecture Reference Manual for A-Profile architecture](#) [3].

D_{BNHQQ} An RTTE has a *state*.

The RTTE state values are shown in the following table.

Name	Description
ASSIGNED	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a DATA Granule.
ASSIGNED_DEV	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a DEV_MAPPED Granule.
ASSIGNED_NS	This RTTE is identified by an Unprotected IPA. The output address of this RTTE points to a Granule-aligned address within NS PAS.
ASSIGNED_VSMMU	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a VSMMU Granule.
AUX_DESTROYED	An auxiliary RTT was destroyed while a corresponding primary RTT entry was live.
TABLE	The output address of this RTTE points to the next-level RTT.
UNASSIGNED	This RTTE is identified by a Protected IPA. This RTTE is not associated with any Granule.
UNASSIGNED_NS	This RTTE is identified by an Unprotected IPA. This RTTE is not associated with any Granule.

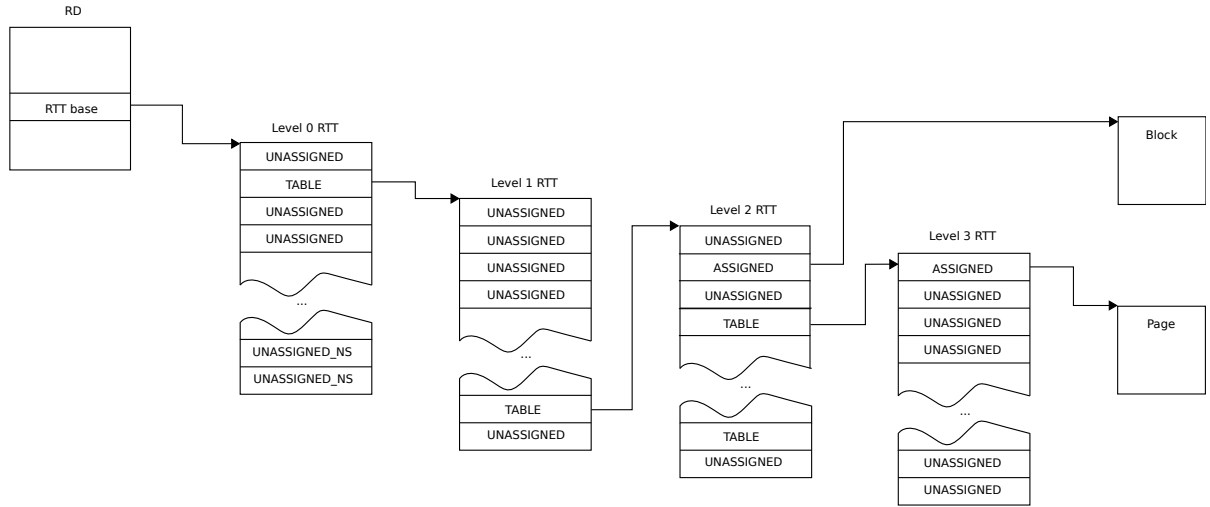
I_{QWQSB} The state of an RTTE in a RTT which is not level 1 or level 2 or level 3 is UNASSIGNED, UNASSIGNED_NS or TABLE.

D_{NSHSL} The output address of an RTTE whose state is TABLE and which is in a level n RTT is the physical address of a level $n+1$ RTT.

I_{DJZTM} An RTT whose level n is not the starting RTT level is pointed-to by exactly one TABLE RTTE in a level $n-1$ RTT.

I_{DXQWZ}

The following diagram shows an example RTT tree, annotated with RTTE states.



I_{FGWQS}

The function `AddrIsRttLevelAligned()` is used to evaluate whether an address is aligned to the address range described by an RTTE at a specified RTT level.

See also:

- [A5.3.1 Host IPA state](#)
- [B1.4 Command condition expressions](#)

A5.6.5 RTT reading

I_{KJWKQ}

Attributes of an RTTE, including the RTTE state, can be read by calling the `RMI_RTT_READ_ENTRY` command. The set of RTTE attributes which are returned depends on the state of the RTTE.

See also:

- [B4.3.44 RMI_RTT_READ_ENTRY command](#)

A5.6.6 RTT folding

D_{RMCLC}

An RTT is *homogeneous* if all of the following are true:

- State of all entries is the same.
- RIPAS of all entries is the same, or the RTT describes Unprotected IPA space.
- S2AP fields of all entries are the same.
- Either the state is UNASSIGNED or UNASSIGNED_NS, or all of the following are true:
 - Level is 2 or 3.
 - Output address of first entry is aligned to size of the address range described by an entry in the parent RTT.
 - Output addresses of all entries are contiguous.
 - Memory attributes of all entries are the same.

I_{KDXLT}

The function `RttIsHomogeneous()` is used to evaluate whether an RTT is homogeneous.

D_{QPCXP}

RTT folding is the operation of destroying a homogeneous child RTT, and moving information which was stored in the child RTT into the parent RTTE.

I_{QMGWK}

On RTT folding, the state of the parent RTTE is determined from the contents of the child RTTEs.

I_{LLWGH}

The function `RttFold()` is used to evaluate the parent RTTE state which results from an RTT folding operation.

I_{TPMGT}

On RTT folding, if the state of the parent RTTE is any of the following then the attributes of the parent RTTE are copied from the child RTTEs:

- ASSIGNED
- ASSIGNED_NS
- ASSIGNED_DEV

See also:

- [A5.6.9 RTT destruction](#)
- [A10.3.2 Stage 2 Access Permissions within a multi-Plane Realm](#)
- [B3.130 RttFold function](#)
- [B3.131 RttIsHomogeneous function](#)
- [B4.3.41 RMI_RTT_FOLD command](#)

A5.6.7 RTT unfolding

D_{HQQMG} *RTT unfolding* is the operation of creating a child RTT, and populating it based on the contents of the parent RTTE.

I_{KWZXN} On RTT unfolding, the state of all RTTEs in the child RTT are set to the state of the parent RTTE.

I_{HMYSW} On RTT unfolding, if the state of the parent RTTE is any of the following then the output addresses of RTTEs in the child RTT are set to a contiguous range which starts from the address of the parent RTTE:

- ASSIGNED
- ASSIGNED_NS
- ASSIGNED_DEV

See also:

- [B4.3.38 RMI_RTT_CREATE command](#)

A5.6.8 RTTE liveness and RTT liveness

D_{KCMLN} *RTTE liveness* is a property which means that a physical address is stored in the RTTE.

D_{HGYJZ} An RTTE is *live* if the RTTE state is any of the following:

- ASSIGNED
- ASSIGNED_NS
- ASSIGNED_DEV
- TABLE
- VSMMU

I_{RHLYZ} The function `RttSkipNonLiveEntries()` is used to scan an RTT to find the next live RTTE. The resulting IPA is returned to the Host from commands whose successful execution causes a live RTTE to become non-live.

X_{GQPSF} Identifying the next live RTTE allows the Host to avoid calls to `RMI_RTT_READ_ENTRY` when unmapping ranges of a Realm's IPA space, for example during Realm destruction.

D_{MPWLR} *RTT liveness* is a property which means that there exists another RMM data structure which is referenced by the RTT.

D_{YPSLW} An RTT is *live* if, for any of its entries, the RTTE state is any of the following:

- ASSIGNED
- ASSIGNED_DEV
- TABLE
- VSMMU

I_{MXJNY} Note that an RTT can be non-live, even if one of its entries is live. This would be the case for example if the RTT corresponds to an Unprotected IPA range and the state of one of its entries is `ASSIGNED_NS`.

I_{YPLKM} The function `RttIsLive()` is used to evaluate whether an RTT is live.

See also:

- [A5.6.9 RTT destruction](#)
- [B3.132 RttIsLive function](#)
- [B3.147 RttSkipNonLiveEntries function](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)
- [B4.3.47 RMI_RTT_UNMAP_UNPROTECTED command](#)

A5.6.9 RTT destruction

D_{VXRZW} *RTT destruction* is the operation of destroying a child RTT, and discarding information which was stored in the child RTT.

I_{PRMFR} An RTT cannot be destroyed if it is live.

I_{MDFQN} An RTT can be destroyed regardless of whether it is homogeneous.

I_{MCKSK} Following RTT destruction, all of the following are true for the parent RTTE:

- RIPAS is DESTROYED
- RTTE state is UNASSIGNED

See also:

- [A5.2 Realm view of memory management](#)
- [A5.6.6 RTT folding](#)
- [A5.6.8 RTTE liveness and RTT liveness](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)

A5.6.10 RTT walk

I_{CBWSX} An IPA is translated to a PA by walking an RTT tree, starting at the RTT base.

I_{FDWYV} The behaviour of an RTT walk is defined by the Armv8-A Virtual Memory System Architecture (VMSA) [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#).

I_{TVGQD} The inputs to an RTT walk are:

- a Realm Descriptor, which contains the address of the initial RTT
- an RTT tree index
- a target IPA
- a target RTT level.

The RTT walk terminates when either:

- it reaches the target RTT level, or
- it reaches an RTTE whose state is not TABLE.

D_{RBHVQ} The result of an RTT walk performed by the RMM is a data structure of type `RmmRttWalkResult`.

The attributes of an `RmmRttWalkResult` are summarized in the following table.

Name	Type	Description
level	Int8	RTT level reached by the walk
rtt_addr	Address	Address of RTT reached by the walk
rtte	RmmRttEntry	RTTE reached by the walk

I_{ZSRCD} The function `RmmRttWalkResult RttWalk(rd, addr, level)` is used to represent an RTT walk.

I_{FBZPQ}

The input address to an RTT walk is always less than 2^w , where w is the IPA width of the target Realm.

See also:

- [A2.1.3 Realm attributes](#)
- [A10.3.1 Auxiliary RTT](#)
- [B1.4 Command condition expressions](#)
- [B3.149 RttWalk function](#)
- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.38 RMI_RTT_CREATE command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)
- [B4.3.43 RMI_RTT_MAP_UNPROTECTED command](#)
- [B4.3.47 RMI_RTT_UNMAP_UNPROTECTED command](#)
- [C2.55 RmmRttWalkResult type](#)

A5.6.11 Stage 2 Access Permissions

This section describes how Stage 2 Access Permissions (S2AP) is managed for Realm IPA space, from the following perspectives:

- How S2AP is encoded in RTT descriptors.
- The programming model for control by the Realm of S2AP for Protected IPA space.
- The programming model for control by the Host of S2AP for Unprotected IPA space.

A5.6.11.1 Encoding of Stage 2 Access Permissions in RTT descriptors

I₀₀₇₂

If the Realm uses S2AP direct encoding then S2AP values are encoded directly in RTT entries.

I₀₀₇₃

If the Realm uses S2AP indirect encoding then S2AP is encoded indirectly in RTT entries, as described in “Stage 2 Indirect permissions” in the [Arm Architecture Reference Manual for A-Profile architecture](#) [3].

See also:

- [Arm Architecture Reference Manual for A-Profile architecture](#) [3]
- [A3.12 Support for Stage 2 Access Permissions indirect encoding](#)

A5.6.11.2 Stage 2 Access Permissions for a Protected IPA

I₀₀₇₄

The programming model for control of S2AP for Protected IPA space is based on indirection, as follows:

- The S2AP base value of a Protected IPA is determined by its HIPAS.
- Each Protected IPA has an *S2AP overlay index* which is controlled by P0, via RSI commands.
- Each { Plane, S2AP overlay index } tuple maps to an *S2AP overlay value*.
 - For Pn, this mapping is controlled by P0 via RSI commands, subject to constraints imposed by the RMM.
 - For P0, this mapping is architecturally fixed.
- The S2AP which applies to an access from a given Plane to a given IPA are defined by the combination of the S2AP base value and the S2AP overlay value, following the rules for “Stage 2 Indirect permissions” in the [Arm Architecture Reference Manual for A-Profile architecture](#) [3].

I₀₀₇₅

The programming model for control of S2AP for Protected IPA space does not depend on whether the Realm uses S2AP indirect encoding.

U₀₀₇₆

If the Realm uses S2AP direct encoding, the RMM maps from the indirect programming model onto S2AP values which are directly stored in RTT entries.

R₀₀₇₇

For a Protected IPA whose HIPAS is ASSIGNED, the S2AP base value is $RW + puX$.

R₀₀₇₈ For a Protected IPA whose HIPAS is ASSIGNED_DEV, the S2AP base value is *RW*.

R₀₀₇₉ For a Protected IPA, all S2AP overlay indices map to *RW+puX* for P0.

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)
- [A10.3.2 Stage 2 Access Permissions within a multi-Plane Realm](#)

A5.6.11.3 Stage 2 Access Permissions for an Unprotected IPA

I₀₀₈₀ The S2AP which applies to a Realm access to an Unprotected IPA is controlled by the Host.

I₀₀₈₁ The programming model for control by the Host of S2AP for Unprotected IPA space depends on whether the Realm uses S2AP indirect encoding.

If the Realm uses S2AP direct encoding then S2AP is controlled as follows:

- The Host provides read and write permissions via the *AP* field in the descriptor passed to *RMI_RTT_MAP_UNPROTECTED* to create the mapping in the primary RTT tree.
- The RMM sets *XN* = '1' in the primary RTT tree.
- Execution of *RMI_RTT_AUX_MAP_UNPROTECTED* causes the *RW* and *XN* fields to be copied from the primary RTT tree into an auxiliary RTT tree.

If the Realm uses S2AP indirect encoding then S2AP is controlled as follows:

- The Host provides an *S2AP base index* in the descriptor passed to *RMI_RTT_MAP_UNPROTECTED*.
- The RMM configures the S2AP overlay index to provide a S2AP overlay value of *RW*.
- The observed S2AP is defined by the combination of the S2AP base value and the S2AP overlay value, following the rules for “Stage 2 Indirect permissions” in the [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#).

In this way, the RMM ensures that neither unprivileged execute permission (*uX*) nor privileged execute permission (*pX*) is observed on Realm access to an Unprotected IPA.

See also:

- [A5.6.11.4 Stage 2 base permission values](#)
- [B4.3.35 RMI_RTT_AUX_MAP_UNPROTECTED command](#)
- [B4.3.43 RMI_RTT_MAP_UNPROTECTED command](#)

A5.6.11.4 Stage 2 base permission values

R₀₀₈₂ The mapping from S2AP base index to S2AP base value is as follows:

S2AP base index	S2AP base value
0	<i>NoAccess</i>
1	<i>RO</i>
2	<i>WO</i>
3	<i>RW</i>
4	<i>RW+puX</i>
5 to 15	Reserved

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)

A5.6.12 Memory attributes

A5.6.12.1 Memory attributes for ASSIGNED mappings

R ₀₀₈₃	The memory type and cacheability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED are Normal Write-Back.
R ₀₀₈₄	The shareability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED are Inner Shareable.
R _{KCFCT}	The memory attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED are independent of any stage 1 descriptors and of the state of the stage 1 MMU.
U _{NPVGN}	The RMM uses FEAT_S2FWB to ensure that the memory attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED are independent of stage 1 translation.

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)
- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)

A5.6.12.2 Memory attributes for ASSIGNED_DEV mappings

R ₀₀₈₅	The memory type and cacheability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED_DEV and which maps to a Device non-coherent memory location are controlled by stage 1 translation and constrained to be one of the following: <ul style="list-style-type: none"> • Device, with device attributes specified via stage 1 translation • Normal Non-Cacheable
U ₀₀₈₆	The RMM uses FEAT_S2FWB to constrain the memory type and cacheability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED_DEV and which is mapped to a Device non-coherent memory physical location.
R ₀₀₈₇	The memory type and cacheability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED_DEV and which maps to a Device coherent memory physical location are passed through from stage 1 translation.
U ₀₀₈₈	The RMM uses FEAT_S2FWB to pass through from stage 1 translation the memory type and cacheability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED_DEV and which is mapped to a Device coherent memory physical location.
R ₀₀₈₉	The shareability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED_DEV and which maps to a Device non-coherent memory physical location are Outer Shareable.
R ₀₀₉₀	The shareability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED_DEV and which maps to a Device coherent memory physical location are Inner Shareable.

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)
- [A2.2.1 Granule category](#)
- [A9.5.3 Realm validation of device memory mappings](#)
- [B4.3.4 RMI_DEV_MEM_MAP command](#)

A5.6.12.3 Memory attributes for ASSIGNED_NS mappings

D _{FJTMF}	The following attributes of an RTT entry whose state is ASSIGNED_NS are <i>Host-controlled Unprotected RTT attributes</i> : <ul style="list-style-type: none"> • ADDR • MemAttr[2:0] • AP (if the Realm uses S2AP direct encoding) or PIIIndex (if the Realm uses S2AP indirect encoding)
--------------------	--

- X_{QHLKB}** In an RTT entry whose state is ASSIGNED_NS, MemAttr[3] is RES0 because the RMM uses FEAT_S2FWB.
- R_{QFLWD}** The shareability attributes which result from Realm access to an IPA whose HIPAS is ASSIGNED_NS are as follows:
- Inner Shareable if the mapping is cacheable.
 - Outer Shareable if the mapping is non-cacheable.
- U_{MCCRT}** The shareability attributes of an RTT entry which corresponds to an Unprotected IPA are expected to be controlled by the RMM as follows:
- If LPA2 is enabled at stage 2 then the RMM is expected to set VTCR_EL2.DS == '1'.
 - If LPA2 is not enabled at stage 2 then the RMM is expected to set the value of the SH field in the translation table descriptor based on the value of the MemAttr field.

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)
- [A3.12 Support for Stage 2 Access Permissions indirect encoding](#)
- [B3.121 RttDescriptorIsValidForUnprotected function](#)
- [B4.3.43 RMI_RTT_MAP_UNPROTECTED command](#)

A5.6.12.4 Summary of memory attributes

I₀₀₉₁ The following table summarizes the resultant memory attributes for each permitted combination of HIPAS and Granule category.

HIPAS	Granule category	Resultant memory type and cacheability attributes	Resultant shareability attributes
ASSIGNED	Delegable DRAM	Normal Write-Back	Inner Shareable
ASSIGNED_DEV	Delegable non-coherent device memory	One of the following: <ul style="list-style-type: none"> • Device • Normal Non-cacheable (Specified by stage 1 translation)	Outer Shareable
ASSIGNED_DEV	Delegable coherent device memory	Specified by stage 1 translation	Inner Shareable
ASSIGNED_NS	Delegable DRAM	Specified by combination of Host-controlled RTT attributes and stage 1 translation	<ul style="list-style-type: none"> • Inner Shareable if the mapping is cacheable • Outer Shareable if the mapping is non-cacheable

See also:

- [A2.2.1 Granule category](#)

A5.6.12.5 Hardware access flag and dirty bit management

- R_{JRZTL}** Hardware access flag and dirty bit management is disabled for the stage 2 translation used by a Realm.
- I_{QFGJC}** Hardware access flag and dirty bit management may be enabled by software executing within the Realm, for its own stage 1 translation.

Chapter A6

Realm interrupts and timers

This specification requires that a virtual Generic Interrupt Controller (vGIC) is presented to a Realm. This vGIC should be architecturally compliant with respect to GICv3 with no legacy operation.

The Host is able to inject virtual interrupts using the GIC virtual CPU interface.

The vGIC presented to a Realm is expected to be implemented via a combination of Host emulation and RMM mediation, as follows:

- Management of Non-secure physical interrupts is performed by the Host, via the GIC Interrupt Routing Infrastructure (IRI).
- The Host is responsible for emulating a GICv3 distributor MMIO interface.
- The Host is responsible for emulating a GICv3 redistributor MMIO interface for each REC.
- The GIC MMIO interfaces emulated by the Host must be presented to the Realm via its Unprotected IPA space.
- The Host may optionally provide a virtual Interrupt Translation Service (ITS). The Realm must allocate ITS tables within its Unprotected IPA space.
- The RMM allows the Host to control some of the GIC virtual CPU interface state which is observed by the Realm. This state is designed to be the minimum required to allow the Host to correctly manage interrupts for the Realm, with integrity guaranteed by the RMM for the remainder of the GIC CPU interface state.
- On REC exit, the RMM exposes some of the GIC virtual CPU interface state to the Host. This state is designed to be the minimum required to allow the Host to correctly manage interrupts for the Realm, with confidentiality guaranteed by the RMM for the remainder of the GIC virtual CPU interface state.

On every REC exit, the EL1 timer state is exposed to the Host. The RMM guarantees that a REC exit occurs whenever a Realm EL1 timer asserts or de-asserts its output.

See also:

- [Arm Generic Interrupt Controller \(GIC\) Architecture Specification version 3 and version 4 \[6\]](#)
- [A5.2.1 Realm IPA space](#)
- [D1.6 Realm interrupts and timers flows](#)

DRAFT

A6.1 Realm interrupts

This section describes the programming model for a REC's GIC CPU interface.

I ₀₀₉₂	The number of GICv3 List Registers which are available is discoverable via the RMI_FEATURES command.
X ₀₀₉₃	Providing an ABI for discovery of the number of GICv3 List Registers which are available allows the RMM to reserve physical GICv3 List Registers for its own usage.
D _{XZVGB}	The value of <code>enter.gicv3_lrs[n]</code> is valid if all of the following are true: <ul style="list-style-type: none"> The value is an architecturally valid encoding of <code>ICH_LR<n>_EL2</code> according to Arm Generic Interrupt Controller (GIC) Architecture Specification version 3 and version 4 [6]. <code>HW == '0'</code>.
X _{DMSDZ}	The GICv3 architecture states that, if <code>HW == '1'</code> then the virtual interrupt must be linked to a physical interrupt whose state is Active, otherwise behavior is undefined. The RMM is unable to validate that invariant, so it imposes the constraint that <code>HW == '0'</code> .
D _{CPLDX}	The value of <code>enter.gicv3_hcr</code> is valid if the value is an architecturally valid encoding of <code>ICH_HCR_EL2</code> according to Arm Generic Interrupt Controller (GIC) Architecture Specification version 3 and version 4 [6] .
R _{HLEFRY}	REC entry fails if the value of any <code>enter.gicv3_*</code> attribute is invalid.
R _{WNFRW}	On REC entry, <code>ICH_LR<n>_EL2</code> is set to <code>enter.gicv3_lrs[n]</code> , for all values of <code>n</code> in the set of GICv3 List Registers which are available.
R _{WVGfJ}	On REC entry, the following fields in <code>ICH_HCR_EL2</code> are set to the corresponding values in <code>enter.gicv3_hcr</code> : <ul style="list-style-type: none"> UIE LRENPIE NPIE VGrp0EIE VGrp0DIE VGrp1EIE VGrp1DIE TDIR
I _{SMHXB}	On REC entry, fields in <code>enter.gicv3_hcr</code> must be set to '0' except for the following: <ul style="list-style-type: none"> UIE LRENPIE NPIE VGrp0EIE VGrp0DIE VGrp1EIE VGrp1DIE TDIR <p>If any other field in <code>enter.gicv3_hcr</code> is set to '1', then <code>RMI_REC_ENTER</code> fails.</p>
X _{LMXCX}	The RMM provides access to the GIC virtual CPU interface to the Realm and therefore controls the enable bit and most trap bits in <code>ICH_HCR_EL2</code> . The maintenance interrupt control bits are controlled by the Host, because the maintenance interrupts are provided as hints to the hypervisor to allocate List Registers optimally and to correctly emulate GICv3 behavior. The <code>TDIR</code> bit is also controlled by the Host because it is used when supporting <code>EOImode == '1'</code> in the Realm. This mode is used to allow deactivation of virtual interrupts across RECs. This deactivation must be handled by the Host because the RMM can only operate on a single REC during execution of <code>RMI_REC_ENTER</code> .
R _{LNQRL}	A REC exit due to IRQ is not generated for an interrupt which is masked by the value of <code>ICC_PMR_EL1</code> at the time of REC entry.

U _{GXCHC}	The RMM should preserve the value of ICC_PMR_EL1 during REC entry.
R _{NKPNC}	On REC exit, <code>exit.gicv3_vmcr</code> contains the value of ICC_VMCR_EL2 at the time of the Realm exit.
R _{SKQNF}	On REC exit, <code>exit.gicv3_misr</code> contains the value of ICC_MISR_EL2 at the time of the Realm exit.
X _{DBGXB}	The Host could in principle infer the value of ICC_MISR_EL2 at the time of the Realm exit from the combination of <code>exit.gicv3_lrs[n]</code> and <code>exit.gicv3_hcr</code> . However, this would be cumbersome, error-prone, and diverge from the design of existing hypervisor software.
R _{QKZXD}	On REC exit, <code>exit.gicv3_lrs[n]</code> contains the value of ICC_LR<n>_EL2 at the time of the Realm exit, for all values of <i>n</i> in the set of GICv3 List Registers which are available.
R _{SNVZH}	On REC exit, the following fields in <code>exit.gicv3_hcr</code> contains the value of the corresponding field in ICC_HCR_EL2 at the time of the Realm exit: <ul style="list-style-type: none"> • EOICount • UIE • LRENPIE • NPIE • VGrp0EIE • VGrp0DIE • VGrp1EIE • VGrp1DIE • TDIR <p>All other fields contain zero.</p>
R _{FGQXT}	On REC exit, the values of the following registers may have changed: <ul style="list-style-type: none"> • ICC_AP0R<n>_EL2 • ICC_AP1R<n>_EL2 • ICC_LR<n>_EL2 • ICC_VMCR_EL2 • ICC_HCR_EL2
S _{QMJJV}	It is the responsibility of the caller to save and restore GIC virtualization system control registers if their value needs to be preserved following execution of RMI_REC_ENTER.
X _{KDGHF}	On REC entry, the values of the GIC virtualization control system registers are overwritten. The Non-secure hypervisor runs at EL2 and therefore does not make direct use of the virtual GIC CPU interface for its own execution. This means that saving / restoring the caller's GIC virtualization control system registers would typically not be required and would add additional runtime overhead for each execution of RMI_REC_ENTER.
R _{VSBBB}	On REC exit, <code>ICC_HCR_EL2.En == '0'</code> .
X _{WLTBX}	Disabling the virtual GIC CPU interface ensures that the caller does not receive unexpected GIC maintenance interrupts. A stronger constraint, for example stating that all GIC virtualization control system registers are zero on REC exit, was considered. However, this was rejected on the basis that it may preclude future optimisations, such as returning early from execution of RMI_REC_ENTER, without needing to first write zero to all GIC virtualization control system registers, if an interrupt is pending.
See also:	
<ul style="list-style-type: none"> • Arm Generic Interrupt Controller (GIC) Architecture Specification version 3 and version 4 [6] • A4.2 REC entry • A4.3 REC exit • A10.4 Planes interrupts • B4.3.6 RMI_FEATURES command • B4.3.30 RMI_REC_ENTER command • B4.4.31 RmiRecEnter type • B4.4.33 RmiRecExit type 	

- [D1.6.1 Interrupt flow](#)

DRAFT

A6.2 Realm timers

This section describes the operation of architectural timers during Realm execution, including the following:

- The behavior of EL2 timers programmed by the Host
- The behavior of EL1 timers as perceived by the Realm
- The Realm timer state which is exposed to the Host on REC exit, in order to facilitate virtualization of timer interrupts

<code>R_{LKNDV}</code>	Architectural timers are available to a Realm and behave according to their architectural specification.
<code>I_{VFYJV}</code>	If the Host has programmed an EL1 timer to assert its output during Realm execution, that timer output is not guaranteed to assert.
<code>R_{FKCHX}</code>	If the Host has programmed an EL2 timer to assert its output during Realm execution, that timer output is guaranteed to assert.
<code>R_{RJZRP}</code>	Both the virtual and physical counter values are guaranteed to be monotonically increasing when read by a Realm, in accordance with the architectural counter behavior.
<code>R_{JSMQP}</code>	A read by a Realm of either the virtual or physical counter at the same place in the instruction flow would return the same value.
<code>X_{YCDMW}</code>	In order to ensure that the Realm has a consistent view of time, the virtual timer offset must be fixed for the lifetime of the Realm. The absolute value of the virtual timer offset is not important, so the value zero has been chosen for simplicity of both the specification and the implementation.
<code>I_{FKMGZ}</code>	The rule that virtual and physical counter values are identical may need to be amended if a future version of the specification supports migration and / or virtualization of time based on the virtual counter differing from the physical counter.
<code>R_{SVCMR}</code>	On a change in the output of an EL1 timer which requires a Realm-observable change to the state of virtual interrupts, a REC exit occurs.
<code>R_{VWQDH}</code>	On REC exit, Realm EL1 timer state is exposed via the <code>RmiRecExit</code> object: <ul style="list-style-type: none"> • <code>exit.cntv_ctl</code> contains the value of <code>CNTV_CTL_EL0</code> at the time of the Realm exit. • <code>exit.cntv_cval</code> contains the value of <code>CNTV_CVAL_EL0</code> at the time of the Realm exit, expressed as if the virtual counter offset was zero. • <code>exit.cntp_ctl</code> contains the value of <code>CNTP_CTL_EL0</code> at the time of the Realm exit. • <code>exit.cntp_cval</code> contains the value of <code>CNTP_CVAL_EL0</code> at the time of the Realm exit, expressed as if the physical counter offset was zero.
<code>S_{PYWWE}</code>	The Host should check the Realm EL1 timer state on every return from <code>RMI_REC_ENTER</code> and update virtual interrupt state accordingly. This is true regardless of the value of <code>exit.exit_reason</code> : even if the return occurred for a reason unrelated to timers (for example, a REC exit due to Data Abort), the Realm EL1 timer state should be checked.
<code>I_{VRWGS}</code>	On REC entry, for both the EL1 Virtual Timer and the EL1 Physical Timer, if the EL1 timer asserts its output in the state described in the REC exit structure from the previous REC exit then the RMM masks the hardware timer signal before returning to the Realm.

This masking is done to allow the Realm to make forward progress, which would otherwise be prevented by the hardware timer generating a physical interrupt that would cause a Realm exit.

During Realm execution, when the hardware timer signal is masked, the Realm may write to the timer registers, causing the hardware timer to become de-asserted and possibly asserted again. Such changes in the output of the EL1 timer are not required to result in a REC exit if the RMM can infer that the change should not result in a Realm-observable change to the state of virtual interrupts.

It is only when a change in the hardware timer output means that the corresponding virtual interrupt needs to be made pending or idle, that a REC exit must occur.

See also:

- [A4.3 REC exit](#)
- [A10.5 Planes timers](#)
- [B4.4.33 RmiRecExit type](#)
- [D1.6.2 Timer interrupt delivery flow](#)

DRAFT

Chapter A7

Realm measurement and attestation

This section describes how the initial state of a Realm is measured and can be attested.

A7.1 Realm measurements

This section describes how Realm measurement values are calculated.

D _{SJWWS}	A Realm measurement value is a rolling hash.
D _{YKDBY}	A <i>Realm Hash Algorithm</i> (RHA) is an algorithm which is used to extend a Realm measurement value.
I _{NRKWB}	The RHA used by a Realm is selected via the <code>hash_algo</code> attribute.

See also:

- [A2.1.3 Realm attributes](#)
- [A3.2 Realm hash algorithm](#)
- [A7.2.3.1.4 Realm Initial Measurement claim](#)
- [A7.2.3.1.5 Realm Extensible Measurements claim](#)

A7.1.1 Realm Initial Measurement

This section describes how the Realm Initial Measurement (RIM) is calculated.

I _{XKSBZ}	The initial RIM value for a Realm is calculated from a subset of the Realm parameters.
I _{NCNDK}	A RIM is extended by applying the RHA to the inputs of RMM operations which are executed during Realm construction.
I _{NQQTf}	The following operations cause a RIM to be extended: <ul style="list-style-type: none">• Creation of a DATA Granule during Realm construction• Creation of a runnable REC• Changes to RIPAS of Protected IPA during Realm construction
R _{VMPZG}	On execution of an operation which requires extension of a RIM, the RMM first constructs a <i>measurement descriptor</i> structure. The measurement descriptor contents include the current RIM value. The new RIM value is computed by applying the RHA to the measurement descriptor.

$$\begin{aligned} desc &= MeasurementDescriptor(M_{i-1}, \dots) \\ M_i &= RHA(desc) \end{aligned}$$

I _{FQHFC}	A RIM is immutable while the state of the Realm is <code>REALM_ACTIVE</code> . This implies that a RIM reflects the configuration and contents of the Realm at the moment when it transitioned from the <code>REALM_NEW</code> to the <code>REALM_ACTIVE</code> state.
I _{DQGPT}	A RIM depends upon the order of the RMM operations which are executed during Realm construction.
S _{VZNCW}	The order in which RMM operations are executed during Realm construction must be agreed between the Realm owner (or a delegate of the Realm owner which will receive and validate the RIM) and the Host which executes the RMM commands. This ensures that a correctly-constructed Realm will have the expected measurement.
I _{LTWBL}	The value of a RIM can be read using the <code>RSI_MEASUREMENT_READ</code> command.

See also:

- [B4.3.1.4 RMI_DATA_CREATE extension of RIM](#)
- [B4.3.25.4 RMI_REALM_CREATE initialization of RIM](#)
- [B4.3.28.4 RMI_REC_CREATE extension of RIM](#)
- [B4.3.42.4 RMI_RTT_INIT_RIPAS extension of RIM](#)
- [B5.3.8 RSI_MEASUREMENT_READ command](#)

A7.1.2 Realm Extensible Measurement

This section describes the behavior of a Realm Extensible Measurement (REM).

I_{QJDDWM}

A REM is extended using the RSI_MEASUREMENT_EXTEND command.

I_{CTMBT}

The value of a REM can be read using the RSI_MEASUREMENT_READ command.

I_{MDQRP}

The initial value of a REM is zero.

See also:

- [B5.3.7 RSI_MEASUREMENT_EXTEND command](#)
- [B5.3.8 RSI_MEASUREMENT_READ command](#)

DRAFT

A7.2 Realm attestation

This section describes the primitives which are used to support remote Realm attestation.

A7.2.1 Attestation token

D_{VRRLN} A CCA attestation token is a collection of claims about the state of a Realm and of the CCA platform on which the Realm is running.

I_{BXSBD} A CCA attestation token consists of two parts:

- Realm token
 - Contains attributes of the Realm, including:
 - Realm Initial Measurement
 - Realm Extensible Measurements
- CCA platform token
 - Contains attributes of the CCA platform on which the Realm is running, including:
 - CCA platform identity
 - CCA platform lifecycle state
 - CCA platform software component measurements

I_{JKJCQ} The size of a CCA attestation token may be greater than 4KB.

See also:

- [A7.1.1 Realm Initial Measurement](#)
- [A7.1.2 Realm Extensible Measurement](#)

A7.2.2 Attestation token generation

I_{KMRH} The process for a Realm to obtain an attestation token is:

- Call `RSI_ATTESTATION_TOKEN_INIT` once
- Call `RSI_ATTESTATION_TOKEN_CONTINUE` in a loop, until the result is not `RSI_INCOMPLETE`

Each call to `RSI_ATTESTATION_TOKEN_CONTINUE` retrieves up to one Granule of the attestation token.

S_{XMLMF}

The following pseudocode illustrates the process of a Realm obtaining an attestation token.

```

int get_attestation_token(...)
{
    int ret;
    uint64_t size, max_size;
    uint64_t buf, granule;

    ret = RSI_ATTESTATION_TOKEN_INIT(challenge, &max_size);
    if (ret) {
        return ret;
    }

    buf = alloc(max_size);
    granule = buf;

    do { // Retrieve one Granule of data per loop iteration
        uint64_t offset = 0;

        do { // Retrieve sub-Granule chunk of data per loop iteration
            size = GRANULE_SIZE - offset;
            ret = RSI_ATTESTATION_TOKEN_CONTINUE(granule, offset, size, &len);
            offset += len;
        } while (ret == RSI_INCOMPLETE && offset < GRANULE_SIZE);

        // "offset" bytes of data are now ready for consumption from "granule"

        if (ret == RSI_INCOMPLETE) {
            granule += GRANULE_SIZE;
        }
    } while ((ret == RSI_INCOMPLETE) && (granule < buf + max_size));

    return ret;
}

```

I_{ZWQCB}

Up to one attestation token generation operation may be ongoing on a REC.

I_{TMJVG}

On execution of RSI_ATTESTATION_TOKEN_INIT, if an attestation token generation operation is ongoing on the calling REC, it is terminated.

I_{WTKDD}

The challenge value provided to RSI_ATTESTATION_TOKEN_INIT is included in the generated attestation token. This allows the relying party to establish freshness of the attestation token.

If the size of the challenge provided by the relying party is less than 64 bytes, it should be zero-padded prior to calling RSI_ATTESTATION_TOKEN_INIT. Arm recommends that the challenge should contain at least 32 bytes of unique data.

I_{GKDJW}

Generation of an attestation token can be a long-running operation, during which interrupts may need to be handled.

I_{CXSJP}

If a physical interrupt becomes pending during execution of RSI_ATTESTATION_TOKEN_CONTINUE, a REC exit due to IRQ can occur.

On the next entry to the REC:

- If a virtual interrupt is pending on that REC, it is taken to the REC's exception handler
- RSI_ATTESTATION_TOKEN_CONTINUE returns RSI_INCOMPLETE
- The REC should call RSI_ATTESTATION_TOKEN_CONTINUE again

See also:

- [A4.3.5 REC exit due to IRQ](#)
- [A6.1 Realm interrupts](#)

- [A7.2.3.1.1 Realm challenge claim](#)
- [B5.3.1 RSI_ATTESTATION_TOKEN_CONTINUE command](#)
- [B5.3.2 RSI_ATTESTATION_TOKEN_INIT command](#)
- [D1.7.1 Attestation token generation flow](#)
- [D1.7.2 Handling interrupts during attestation token generation flow](#)

A7.2.3 Attestation token format

<code>I_TFHGX</code>	The CCA attestation token is a profiled IETF Entity Attestation Token (EAT).
<code>I_LPTVH</code>	The CCA attestation token is a Concise Binary Object Representation (CBOR) map, in which the map values are the Realm token and the CCA platform token.
<code>I_YZPHG</code>	The Realm token contains structured data in CBOR, wrapped with a COSE_Sign1 envelope according to the CBOR Object Signing and Encryption (COSE) standard.
<code>I_MMQZG</code>	The Realm token is signed by the Realm Attestation Key (RAK).
<code>I_WBGNP</code>	The CCA platform token contains structured data in CBOR, wrapped with a COSE_Sign1 envelope according to the COSE standard.
<code>I_CGYKX</code>	The CCA platform token is signed by the Initial Attestation Key (IAK).
<code>I_CCGQH</code>	The CCA platform token contains a hash of RAK_pub. This establishes a cryptographic binding between the Realm token and the CCA platform token.
<code>I_PTKYD</code>	<p>The CCA attestation token is defined as follows:</p> <hr/> <pre>cca-token = #6.399(cca-token-collection) ; CMW Collection ; (draft-ietf-rats-msg-wrap) cca-platform-token = bstr .cbor COSE_Sign1_Tagged cca-realm-delegated-token = bstr .cbor COSE_Sign1_Tagged cca-token-collection = { 44234 => cca-platform-token ; 44234 = 0xACCA 44241 => cca-realm-delegated-token } ; EAT standard definitions COSE_Sign1_Tagged = #6.18(COSE_Sign1) ; Deliberately shortcut these definitions until EAT is finalised and able to ; pull in the full set of definitions COSE_Sign1 = "COSE-Sign1 placeholder"</pre> <hr/>
<code>I_HZNNH</code>	The composition of the CCA attestation token is summarised in the following figure.

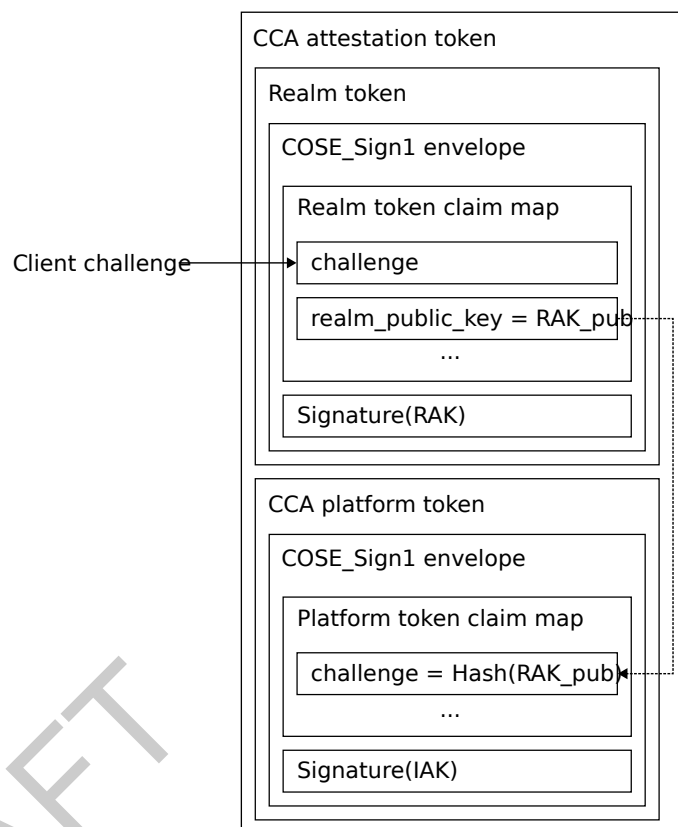


Figure A7.1: Attestation token format

See also:

- [Arm CCA Security model](#) [4]
- [Concise Binary Object Representation \(CBOR\)](#) [7]
- [CBOR Object Signing and Encryption \(COSE\)](#) [8]
- [Entity Attestation Token \(EAT\)](#) [9]
- [A7.2.3.1 Realm claims](#)
- [A7.2.3.2 CCA platform claims](#)

A7.2.3.1 Realm claims

This section defines the format of the Realm token claim map. The format is described using a combination of Concise Data Definition Language (CDDL) and text description.

I_{HKBHC}

The Realm token claim map is defined as follows:

```
cca-realm-claims = (cca-realm-claim-map)

cca-realm-claim-map = {
    cca-realm-challenge
    ? cca-realm-profile
    cca-realm-personalization-value
    cca-realm-initial-measurement
    cca-realm-extensible-measurements
    cca-realm-hash-algo-id
    cca-realm-public-key
    cca-realm-public-key-hash-algo-id
    cca-realm-mec-policy
}
```

See also:

- [Concise Data Definition Language \(CDDL\) \[10\]](#)
- [A7.2.3.1.1 Realm challenge claim](#)
- [A7.2.3.1.2 Realm profile claim](#)
- [A7.2.3.1.3 Realm Personalization Value claim](#)
- [A7.2.3.1.4 Realm Initial Measurement claim](#)
- [A7.2.3.1.5 Realm Extensible Measurements claim](#)
- [A7.2.3.1.6 Realm hash algorithm ID claim](#)
- [A7.2.3.1.7 Realm MEC policy claim](#)
- [A7.2.3.1.8 Realm public key claim](#)
- [A7.2.3.1.9 Realm public key hash algorithm identifier claim](#)
- [A7.2.3.1.10 Collated CDDL for Realm claims](#)
- [A7.2.3.1.11 Example Realm claims](#)

A7.2.3.1.1 Realm challenge claim

I_{TFWXQ}

The Realm challenge claim is used to carry the challenge provided by the caller to demonstrate freshness of the generated token.

I_{RVLZK}

The Realm challenge claim is identified using the EAT_{nonce} label (10).

I_{MNVNP}

The length of the Realm challenge is 64 bytes.

I_{PXMXF}

The Realm challenge claim must be present in a Realm token.

I_{BXGFN}

The format of the Realm challenge claim is defined as follows:

```
cca-realm-challenge-label = 10
cca-realm-challenge-type = bytes .size 64

cca-realm-challenge = (
    cca-realm-challenge-label => cca-realm-challenge-type
)
```

See also:

- [A7.2.2 Attestation token generation](#)
- [B5.3.2 RSI_ATTESTATION_TOKEN_INIT command](#)

A7.2.3.1.2 Realm profile claim

I _{CVNNV}	The Realm profile claim identifies the EAT profile to which the Realm token conforms.
I _{SMSCF}	The Realm profile claim is identified using the EAT _{profile} label (265).
I _{XSSJY}	The Realm profile claim is optional in a CCA Realm token.
I _{GQTJT}	If the Realm profile is not included in a CCA Realm token then the profile value used in the CCA Platform token should refer to a profile that describes both Platform and Realm claims.
I _{SWDJM}	The format of the Realm profile claim is defined as follows:

```
cca-realm-profile-label = 265 ; EAT profile

cca-realm-profile-type = "tag:arm.com,2024:realm#1.1.0"

cca-realm-profile = (
    cca-realm-profile-label => cca-realm-profile-type
)
```

A7.2.3.1.3 Realm Personalization Value claim

I _{SCNXB}	The Realm Personalization Value claim contains the RPV which was provided at Realm creation.
I _{BKZPD}	The Realm Personalization Value claim must be present in a Realm token.
I _{QKNDV}	The format of the Realm Personalization Value claim is defined as follows:

```
cca-realm-personalization-value-label = 44235
cca-realm-personalization-value-type = bytes .size 64

cca-realm-personalization-value = (
    cca-realm-personalization-value-label => cca-realm-personalization-value-type
)
```

See also:

- [A2.1.3 Realm attributes](#)

A7.2.3.1.4 Realm Initial Measurement claim

I _{BXKGD}	The Realm Initial Measurement claim contains the values of the Realm Initial Measurement.
I _{FZQSM}	The Realm Initial Measurement claim must be present in a Realm token.
I _{GGTNH}	The format of the Realm Initial Measurement claim is defined as follows:

```
cca-realm-measurement-type = bytes .size 32 / bytes .size 48 / bytes .size 64
cca-realm-initial-measurement-label = 44238

cca-realm-initial-measurement = (
    cca-realm-initial-measurement-label => cca-realm-measurement-type
)
```

See also:

- [A7.1 Realm measurements](#)
- [A7.2.3.1.5 Realm Extensible Measurements claim](#)

A7.2.3.1.5 Realm Extensible Measurements claim

I _{KFNMV}	The Realm Extensible Measurements claim contains the values of the Realm Extensible Measurements.
I _{DSNFB}	The Realm Extensible Measurements claim must be present in a Realm token.

I_{ZKVMN}

The format of the Realm measurements claim is defined as follows:

```
cca-realm-measurement-type = bytes .size 32 / bytes .size 48 / bytes .size 64
cca-realm-extensible-measurements-label = 44239

cca-realm-extensible-measurements = (
    cca-realm-extensible-measurements-label => [ 4*4 cca-realm-measurement-type ]
)
```

See also:

- [A7.1 Realm measurements](#)
- [A7.2.3.1.4 Realm Initial Measurement claim](#)

A7.2.3.1.6 Realm hash algorithm ID claim

I_{DGCGG}

The Realm hash algorithm ID claim identifies the algorithm used to calculate all hash values which are present in the Realm token.

I_{PVLCJ}

Arm recommends that the value of the Realm hash algorithm ID claim is an IANA Hash Function name [IANA Named Information Hash Algorithm Registry](#) [11].

I_{WKVCQ}

The Realm hash algorithm ID claim must be present in a Realm token.

I_{PWPLJ}

The format of the Realm hash algorithm ID claim is defined as follows:

```
cca-realm-hash-algo-id-label = 44236

cca-realm-hash-algo-id = (
    cca-realm-hash-algo-id-label => text
)
```

A7.2.3.1.7 Realm MEC policy claim

I₀₀₉₄

The Realm MEC policy identifies the MEC policy of the Realm.

I₀₀₉₅

The Realm MEC policy claim must be present in a Realm token.

R₀₀₉₆

On a platform which does not implement FEAT_MEC, the value of the Realm MEC policy claim is “shared”.

I₀₀₉₇

The format of the Realm MEC policy claim is defined as follows:

```
cca-realm-mec-policy-label = 44241

cca-realm-mec-policy = (
    cca-realm-mec-policy-label => "shared" / "private"
)
```

See also:

- [Chapter A11 Realm memory encryption](#)

A7.2.3.1.8 Realm public key claim

I_{ZCFMQ}

The Realm public key claim identifies the key which is used to sign the Realm token.

I_{WBNHC}

The value of the Realm public key claim is a CBOR bstr of a COSE_Key structure. The parameters used for the COSE_Key are profile-specific.

I_{LSNPQ}

The Realm public key claim must be present in a Realm token.

I_{NNNDS}

The format of the Realm public key claim is defined as follows:

```
cca-realm-public-key-label = 44237

cca-realm-public-key-type = bstr .cbor COSE_Key
```

```

cca-realm-public-key = (
    cca-realm-public-key-label => cca-realm-public-key-type
)
COSE_Key-label = int / tstr

COSE_Key-values = any

; See RFC8152 for full definition of COSE_Key
COSE_Key = {
    1 => tstr / int,          ; kty
    ? 2 => bstr,              ; kid
    ? 3 => tstr / int,        ; alg
    ? 4 => [+ (tstr / int) ], ; key_ops
    ? 5 => bstr,              ; Base IV
    * COSE_Key-label => COSE_Key-values
}

```

See also:

- [SEC 1: Elliptic Curve Cryptography, version 2.0 \[12\]](#)
- [A7.2.3.1.9 Realm public key hash algorithm identifier claim](#)
- [A7.2.3.2.2 CCA platform challenge claim](#)

A7.2.3.1.9 Realm public key hash algorithm identifier claim

I_{WWSLP}

The Realm public key hash algorithm identifier claim identifies the algorithm used to calculate H(RAK_{pub}).

I_{TNRBN}

The Realm public key hash algorithm identifier claim must be present in a Realm token.

I_{NNPVX}

The format of the Realm public key hash algorithm identifier claim is defined as follows:

```

cca-realm-public-key-hash-algo-id-label = 44240

cca-realm-public-key-hash-algo-id = (
    cca-realm-public-key-hash-algo-id-label => text
)

```

See also:

- [SEC 1: Elliptic Curve Cryptography, version 2.0 \[12\]](#)
- [A7.2.3.1.8 Realm public key claim](#)
- [A7.2.3.2.2 CCA platform challenge claim](#)

A7.2.3.1.10 Collated CDDL for Realm claims

D_{DCYXZ}

The format of the Realm token claim map is defined as follows:

```
cca-realm-claims = (cca-realm-claim-map)

cca-realm-claim-map = {
    cca-realm-challenge
    ? cca-realm-profile
    cca-realm-personalization-value
    cca-realm-initial-measurement
    cca-realm-extensible-measurements
    cca-realm-hash-algo-id
    cca-realm-public-key
    cca-realm-public-key-hash-algo-id
    cca-realm-mec-policy
}
cca-realm-challenge-label = 10
cca-realm-challenge-type = bytes .size 64

cca-realm-challenge = (
    cca-realm-challenge-label => cca-realm-challenge-type
)
cca-realm-profile-label = 265 ; EAT profile

cca-realm-profile-type = "tag:arm.com,2024:realm#1.1.0"

cca-realm-profile = (
    cca-realm-profile-label => cca-realm-profile-type
)
cca-realm-personalization-value-label = 44235
cca-realm-personalization-value-type = bytes .size 64

cca-realm-personalization-value = (
    cca-realm-personalization-value-label => cca-realm-personalization-value-type
)
cca-realm-measurement-type = bytes .size 32 / bytes .size 48 / bytes .size 64
cca-realm-initial-measurement-label = 44238

cca-realm-initial-measurement = (
    cca-realm-initial-measurement-label => cca-realm-measurement-type
)
cca-realm-extensible-measurements-label = 44239

cca-realm-extensible-measurements = (
    cca-realm-extensible-measurements-label => [ 4*4 cca-realm-measurement-type ]
)
cca-realm-hash-algo-id-label = 44236

cca-realm-hash-algo-id = (
    cca-realm-hash-algo-id-label => text
)
cca-realm-public-key-label = 44237

cca-realm-public-key-type = bstr .cbor COSE_Key

cca-realm-public-key = (
    cca-realm-public-key-label => cca-realm-public-key-type
)
COSE_Key-label = int / tstr
```

```
COSE_Key-values = any

; See RFC8152 for full definition of COSE_Key
COSE_Key = {
    1 => tstr / int,          ; kty
    ? 2 => bstr,              ; kid
    ? 3 => tstr / int,        ; alg
    ? 4 => [+ (tstr / int) ], ; key_ops
    ? 5 => bstr,              ; Base IV
    * COSE_Key-label => COSE_Key-values
}
cca-realm-public-key-hash-algo-id-label = 44240

cca-realm-public-key-hash-algo-id = (
    cca-realm-public-key-hash-algo-id-label => text
)
cca-realm-mec-policy-label = 44241

cca-realm-mec-policy = (
    cca-realm-mec-policy-label => "shared" / "private"
)
```

DRAFT

A7.2.3.1.11 Example Realm claims

 $\mathbb{I}_{\text{CPTFR}}$

An example Realm claim map is shown below in COSE-DIAG format:

[illegible]

A7.2.3.2 CCA platform claims

This section defines the format of the CCA platform token claim map. The format is described using a combination of Concise Data Definition Language (CDDL) and text description.

I_{FJKFY}

The CCA platform token claim map is defined as follows:

```
cca-platform-claims = (cca-platform-claim-map)

cca-platform-claim-map = {
    cca-platform-profile
    cca-platform-challenge
    cca-platform-implementation-id
    cca-platform-instance-id
    cca-platform-config
    cca-platform-lifecycle
    cca-platform-sw-components
    ? cca-platform-verification-service
    cca-platform-hash-algo-id
}
```

See also:

- [Concise Data Definition Language \(CDDL\) \[10\]](#)
- [A7.2.3.2.1 CCA platform profile claim](#)
- [A7.2.3.2.2 CCA platform challenge claim](#)
- [A7.2.3.2.3 CCA platform Implementation ID claim](#)
- [A7.2.3.2.4 CCA platform Instance ID claim](#)
- [A7.2.3.2.5 CCA platform config claim](#)
- [A7.2.3.2.6 CCA platform lifecycle claim](#)
- [A7.2.3.2.7 CCA platform software components claim](#)
- [A7.2.3.2.8 CCA platform verification service claim](#)
- [A7.2.3.2.9 CCA platform hash algorithm ID claim](#)
- [A7.2.3.3.1 Collated CDDL for CCA platform claims](#)
- [A7.2.3.3.2 Example CCA platform claims](#)

A7.2.3.2.1 CCA platform profile claim

I_{FQYTP}

The CCA platform profile claim identifies the EAT profile to which the CCA platform token conforms. Note that because the platform token is expected to be issued when bound to a Realm token, the profile document should also include the relevant Realm profile or a reference to that profile.

I_{XMVFR}

The CCA platform profile claim is identified using the EAT `profile` label (265).

I_{GKKNR}

The CCA platform profile claim must be present in a CCA platform token.

I_{MHRTD}

The format of the CCA platform profile claim is defined as follows:

```
cca-platform-profile-label = 265 ; EAT profile

cca-platform-profile-type = "tag:arm.com,2024:cca_platform#1.1.0"

cca-platform-profile = (
    cca-platform-profile-label => cca-platform-profile-type
)
```

A7.2.3.2.2 CCA platform challenge claim

I_{TKTWZ}

The CCA platform challenge claim contains a hash of the public key used to sign the Realm token.

I_{CLJJK}

The CCA platform challenge claim is identified using the EAT `nonce` label (10).

I_{XHLVJ}

The length of the CCA platform challenge is either 32, 48 or 64 bytes.

I_{GVHNX} The CCA platform challenge claim must be present in a CCA platform token.

I_{LRWHR} The format of the CCA platform challenge claim is defined as follows:

```
cca-hash-type = bytes .size 32 / bytes .size 48 / bytes .size 64
cca-platform-challenge-label = 10
```

```
cca-platform-challenge = (
    cca-platform-challenge-label => cca-hash-type
)
```

See also:

- [A7.2.3.1.8 Realm public key claim](#)

A7.2.3.2.3 CCA platform Implementation ID claim

I_{SMWND} The CCA platform Implementation ID claim uniquely identifies the implementation of the CCA platform.

I_{NDVFB} The value of the CCA platform Implementation ID claim can be used by a verification service to locate the details of the CCA platform implementation from an endorser or manufacturer. Such details are used by a verification service to determine the security properties or certification status of the CCA platform implementation.

I_{RXPVW} The semantics of the CCA platform Implementation ID value are defined by the manufacturer or a particular certification scheme. For example, the ID could take the form of a product serial number, database ID, or other appropriate identifier.

I_{SRPZY} The CCA platform Implementation ID claim does not identify a particular instance of the CCA implementation.

I_{NTCFY} The CCA platform Implementation ID claim must be present in a CCA platform token.

I_{DHYDG} The format of the CCA platform Implementation ID claim is defined as follows:

```
cca-platform-implementation-id-label = 2396 ; PSA implementation ID
cca-platform-implementation-id-type = bytes .size 32

cca-platform-implementation-id = (
    cca-platform-implementation-id-label => cca-platform-implementation-id-type
)
```

See also:

- [Arm CCA Security model \[4\]](#)
- [A7.2.3.2.4 CCA platform Instance ID claim](#)

A7.2.3.2.4 CCA platform Instance ID claim

I_{ZYZB} The CCA platform Instance ID claim represents the unique identifier of the Initial Attestation Key (IAK) for the CCA platform.

I_{XVLLN} The CCA platform Instance ID claim is identified using the EAT ueid label (256).

R_{HVTNC} The first byte of the CCA platform Instance ID value must be 0x01.

I_{ZNGDF} The CCA platform Instance ID claim must be present in a CCA platform token.

I_{VPKJN} The format of the CCA platform Instance ID claim is defined as follows:

```
cca-platform-instance-id-label = 256 ; EAT ueid

; TODO: require that the first byte of cca-platform-instance-id-type is 0x01
; EAT UEIDs need to be 7 - 33 bytes
cca-platform-instance-id-type = bytes .size 33

cca-platform-instance-id = (
    cca-platform-instance-id-label => cca-platform-instance-id-type
)
```

See also:

- [Arm CCA Security model \[4\]](#)
- [A7.2.3.2.3 CCA platform Implementation ID claim](#)

A7.2.3.2.5 CCA platform config claim

<code>I_WVQJT</code>	The CCA platform config claim describes the set of chosen implementation options of the CCA platform. As an example, these may include a description of the level of physical memory protection which is provided.
<code>U_GPXWX</code>	The CCA platform config claim is expected to contain the System Properties field which is present in the Root Non-volatile Storage (RNVS) public parameters.
<code>I_MJHQJ</code>	The CCA platform config claim must be present in a CCA platform token.

```
cca-platform-config-label = 2401 ; PSA platform range
                                ; TBD: add to IANA registration
cca-platform-config-type = bytes

cca-platform-config = (
    cca-platform-config-label => cca-platform-config-type
)
```

See also:

- [RME system architecture spec \[13\]](#)

A7.2.3.2.6 CCA platform lifecycle claim

<code>I_SYKFY</code>	The CCA platform lifecycle claim identifies the lifecycle state of the CCA platform.
<code>R_NBFVV</code>	The value of the CCA platform lifecycle claim is an integer which is divided as follows: <ul style="list-style-type: none"> • value[15:8]: CCA platform lifecycle state • value[7:0]: IMPLEMENTATION DEFINED
<code>I_WFZHV</code>	The CCA platform lifecycle claim must be present in a CCA platform token.
<code>I_QFYLF</code>	A non debugged CCA platform will be in psa-lifecycle-secured state. Realm Management Security Domain debug is always recoverable, and would therefore be represented by psa-lifecycle-non-psa-rot-debug state. Root world debug is recoverable on a HES system and would be represented by psa-lifecycle-recoverable-psa-rot state. On a non-HES system Root world debug is usually non-recoverable, and would be represented by psa-lifecycle-lifecycle-decommissioned state.
<code>I_HMZLL</code>	The format of the CCA platform lifecycle claim is defined as follows:

```
cca-platform-lifecycle-label = 2395 ; PSA lifecycle

cca-platform-lifecycle-unknown-type = 0x0000..0x00ff
cca-platform-lifecycle-assembly-and-test-type = 0x1000..0x10ff
cca-platform-lifecycle-cca-platform-rot-provisioning-type = 0x2000..0x20ff
cca-platform-lifecycle-secured-type = 0x3000..0x30ff
cca-platform-lifecycle-non-cca-platform-rot-debug-type = 0x4000..0x40ff
cca-platform-lifecycle-recoverable-cca-platform-rot-debug-type = 0x5000..0x50ff
cca-platform-lifecycle-decommissioned-type = 0x6000..0x60ff

cca-platform-lifecycle-type =
    cca-platform-lifecycle-unknown-type /
    cca-platform-lifecycle-assembly-and-test-type /
    cca-platform-lifecycle-cca-platform-rot-provisioning-type /
    cca-platform-lifecycle-secured-type /
    cca-platform-lifecycle-non-cca-platform-rot-debug-type /
    cca-platform-lifecycle-recoverable-cca-platform-rot-debug-type /
```

```
cca-platform-lifecycle-decommissioned-type

cca-platform-lifecycle = (
    cca-platform-lifecycle-label => cca-platform-lifecycle-type
)
```

See also:

- [Arm CCA Security model \[4\]](#)

A7.2.3.2.7 CCA platform software components claim

I_{PJCSC} The CCA platform software components claim is a list of software components which can affect the behavior of the CCA platform. It is expected that an implementation will describe the expected software component values within the profile.

U₀₀₉₈ In some implementations, a software component may consist of a configuration data item.

I_{TJTXG} The CCA platform software components claim must be present in a CCA platform token.

I_{DPSKT} The format of the CCA platform software components claim is defined as follows:

```
cca-platform-sw-components-label = 2399 ; PSA software components

cca-platform-sw-component = {
    ? 1 => text,                ; component type
    2 => cca-hash-type,         ; measurement value
    ? 4 => text,                ; version
    5 => cca-hash-type,         ; signer id
    ? 6 => text,                ; hash algorithm identifier
    ? 7 => bool,                ; live firmware activation supported
    ? 8 => [ + cca-hash-type ], ; list of countersigner ids
}

cca-platform-sw-components = (
    cca-platform-sw-components-label => [ + cca-platform-sw-component ]
)
```

CCA platform software component type

I_{PDNCF} The CCA platform software component type is a string which represents the role of the software component.

I_{TPSYF} The CCA platform software component type is intended for use as a hint to help the relying party understand how to evaluate the CCA platform software component measurement value.

R_{RSNBH} The CCA platform software component type is optional in a CCA platform token.

U₀₀₉₉ If the CCA platform supports Live Firmware Activation, one entry in the platform software component table is reserved to act as a measurement register for the Firmware Activity Log. This entry is identified by having a software component type of “FAL”.

See also:

- [A3.13 Live Firmware Activation](#)

CCA platform software component measurement value

I_{RWDKD} The CCA platform software component measurement value represents a hash of the state of the software component in memory at the time it was initialized.

R_{TVXRZ} The CCA platform software component measurement value must be a hash of 256 bits or stronger.

R_{LGBCM} The CCA platform software component measurement value must be present in a CCA platform token.

CCA platform software component version

I_{JVJFW} The CCA platform software component version is a text string whose meaning is defined by the software component vendor.

R_{CZRXB} The CCA platform software component version is optional in a CCA platform token.

CCA platform software component signer ID

I_{DCDMR} The CCA platform software component signer ID is the hash of a signing authority public key for the software component. It can be used by a verifier to ensure that the software component was signed by an expected trusted source.

R_{PXRMC} The CCA platform software component signer ID value must be a hash of 256 bits or stronger.

R_{XPHQC} The CCA platform software signer ID must be present in a CCA platform token.

CCA platform software component hash algorithm ID

I_{TQWZX} The CCA platform software component hash algorithm ID identifies the way in which the hash algorithm used to measure the CCA platform software component.

I_{HHBHG} Arm recommends that the value of the CCA platform software component hash algorithm ID is an IANA Hash Function name [IANA Named Information Hash Algorithm Registry](#) [11].

I_{NJYCM} Arm recommends that the hash algorithm used to measure the CCA platform software component is one of the algorithms listed in the [Arm CCA Security model](#) [4].

I_{HPHCD} The CCA platform software component hash algorithm ID is optional in a CCA platform token.

CCA platform software component Live Firmware Activation support

I₀₁₀₀ The CCA platform software component Live Firmware Activation support attribute declares whether an individual component is subject to Live Firmware Activation. If the attribute is False, the component will not be updated before the next CCA platform reset.

I₀₁₀₁ The CCA platform software component Live Firmware activation support attribute is optional in a CCA platform token.

See also:

- [A3.13 Live Firmware Activation](#)

CCA platform software component countersigner ID list

I₀₁₀₂ The CCA platform software component countersigner ID list contains hashes of public keys which identify signing authorities that provides additional trustworthiness information for the software component. These signatures are provided in addition to the primary signature, which is identified by the CCA platform software component signer ID.

U₀₁₀₃ Example use cases for CCA platform software component countersignatures include:

- An indication of approval for the component, provided by the owner of the CCA platform
- An indication of approval for the component, provided by a third party auditor

U₀₁₀₄ The order of multiple entries within the countersigner ID list may imply a hierarchy. The existence and meaning of any such hierarchy is IMPLEMENTATION DEFINED.

I₀₁₀₅ The CCA platform software component countersigner ID list is optional in a CCA platform token.

A7.2.3.2.8 CCA platform verification service claim

I_{NSTDP} The CCA platform verification service claim is a hint which can be used by a relying party to locate a verifier for the token.

I_{RZJSQ} The value of the CCA platform verification service claim is a text string which can be used to locate the service or a URL specifying the address of the service.

I_{MFYCX} The CCA platform verification service claim may be ignored by a relying party in favor of other information.

I_{MRSXY} The CCA platform verification service claim is optional in a CCA platform token.

I_{WRJSX} The format of the CCA platform verification service claim is defined as follows:

```
cca-platform-verification-service-label = 2400 ; PSA verification service
cca-platform-verification-service-type = text

cca-platform-verification-service = (
    cca-platform-verification-service-label =>
    cca-platform-verification-service-type
)
```

A7.2.3.2.9 CCA platform hash algorithm ID claim

I_{VDZMF} The CCA platform hash algorithm ID claim identifies the default algorithm used to calculate measurements in the CCA platform token.

I_{XHJFX} The default hash algorithm may be overridden for an individual software component, by the CCA platform software component hash algorithm ID claim.

I_{YRPYY} Arm recommends that the value of the CCA platform hash algorithm ID claim is an IANA Hash Function name [IANA Named Information Hash Algorithm Registry \[11\]](#).

I_{TQSTK} The CCA platform hash algorithm ID claim must be present in a CCA platform token.

I_{RKZJT} The format of the CCA platform hash algorithm ID claim is defined as follows:

```
cca-platform-hash-algo-id-label = 2402 ; PSA platform range
                                   ; TBD: add to IANA registration

cca-platform-hash-algo-id = (
    cca-platform-hash-algo-id-label => text
)
```

A7.2.3.3 Attestation token format compatibility

I₀₁₀₆ The table below summarises the support for each claim, depending on the version of the profile of the attestation token. The profile version is reported in the cca-platform-profile claim as <http://arm.com/CCA-SSD/VERSION>.

Claim	Version	
	1.0.0	1.1.0
cca-platform-profile	Required	Required
cca-platform-challenge	Required	Required
cca-platform-implementation-id	Required	Required
cca-platform-instance-id	Required	Required
cca-platform-config	Required	Required
cca-platform-lifecycle	Required	Required

Claim	Version	
	1.0.0	1.1.0
cca-platform-sw-components	Required: <ul style="list-style-type: none"> • Measurement value • Signer ID Optional: <ul style="list-style-type: none"> • Type • Version • Hash algorithm identifier 	Required <ul style="list-style-type: none"> • Measurement value • Signer ID Optional: <ul style="list-style-type: none"> • Type • Version • Hash algorithm identifier • LFA supported • Countersigner IDs
cca-platform-verification-service	Optional	Optional
cca-platform-hash-algo-id	Required	Required
cca-realm-challenge	Required	Required
cca-realm-personalization-value	Required	Required
cca-realm-initial-measurement	Required	Required
cca-realm-extensible-measurements	Required	Required
cca-realm-hash-algo-id	Required	Required
cca-realm-mec-policy	Not supported	Required
cca-realm-public-key	Required	Required
cca-realm-public-key-hash-algo-id	Required	Required

A7.2.3.3.1 Collated CDDL for CCA platform claims

D_{DVMJZ}

The format of the CCA platform token claim map is defined as follows:

```
cca-platform-claims = (cca-platform-claim-map)

cca-platform-claim-map = {
    cca-platform-profile
    cca-platform-challenge
    cca-platform-implementation-id
    cca-platform-instance-id
    cca-platform-config
    cca-platform-lifecycle
    cca-platform-sw-components
    ? cca-platform-verification-service
    cca-platform-hash-algo-id
}
cca-platform-profile-label = 265 ; EAT profile

cca-platform-profile-type = "tag:arm.com,2024:cca_platform#1.1.0"

cca-platform-profile = (
    cca-platform-profile-label => cca-platform-profile-type
)
cca-hash-type = bytes .size 32 / bytes .size 48 / bytes .size 64
cca-platform-challenge-label = 10

cca-platform-challenge = (
    cca-platform-challenge-label => cca-hash-type
)
cca-platform-implementation-id-label = 2396 ; PSA implementation ID
cca-platform-implementation-id-type = bytes .size 32

cca-platform-implementation-id = (
    cca-platform-implementation-id-label => cca-platform-implementation-id-type
)
cca-platform-instance-id-label = 256 ; EAT uuid

; TODO: require that the first byte of cca-platform-instance-id-type is 0x01
; EAT UEIDs need to be 7 - 33 bytes
cca-platform-instance-id-type = bytes .size 33

cca-platform-instance-id = (
    cca-platform-instance-id-label => cca-platform-instance-id-type
)
cca-platform-config-label = 2401 ; PSA platform range
                                ; TBD: add to IANA registration
cca-platform-config-type = bytes

cca-platform-config = (
    cca-platform-config-label => cca-platform-config-type
)
cca-platform-lifecycle-label = 2395 ; PSA lifecycle

cca-platform-lifecycle-unknown-type = 0x0000..0x00ff
cca-platform-lifecycle-assembly-and-test-type = 0x1000..0x10ff
cca-platform-lifecycle-cca-platform-rot-provisioning-type = 0x2000..0x20ff
cca-platform-lifecycle-secured-type = 0x3000..0x30ff
cca-platform-lifecycle-non-cca-platform-rot-debug-type = 0x4000..0x40ff
cca-platform-lifecycle-recoverable-cca-platform-rot-debug-type = 0x5000..0x50ff
cca-platform-lifecycle-decommissioned-type = 0x6000..0x60ff
```

```
cca-platform-lifecycle-type =  
    cca-platform-lifecycle-unknown-type /  
    cca-platform-lifecycle-assembly-and-test-type /  
    cca-platform-lifecycle-cca-platform-rot-provisioning-type /  
    cca-platform-lifecycle-secured-type /  
    cca-platform-lifecycle-non-cca-platform-rot-debug-type /  
    cca-platform-lifecycle-recoverable-cca-platform-rot-debug-type /  
    cca-platform-lifecycle-decommissioned-type  
  
cca-platform-lifecycle = (  
    cca-platform-lifecycle-label => cca-platform-lifecycle-type  
)  
cca-platform-sw-components-label = 2399 ; PSA software components  
  
cca-platform-sw-component = {  
    ? 1 => text,                ; component type  
    ? 2 => cca-hash-type,       ; measurement value  
    ? 4 => text,                ; version  
    ? 5 => cca-hash-type,       ; signer id  
    ? 6 => text,                ; hash algorithm identifier  
    ? 7 => bool,                ; live firmware activation supported  
    ? 8 => [ + cca-hash-type ], ; list of countersigner ids  
}  
  
cca-platform-sw-components = (  
    cca-platform-sw-components-label => [ + cca-platform-sw-component ]  
)  
cca-platform-verification-service-label = 2400 ; PSA verification service  
cca-platform-verification-service-type = text  
  
cca-platform-verification-service = (  
    cca-platform-verification-service-label =>  
    cca-platform-verification-service-type  
)  
cca-platform-hash-algo-id-label = 2402 ; PSA platform range  
                                     ; TBD: add to IANA registration  
  
cca-platform-hash-algo-id = (  
    cca-platform-hash-algo-id-label => text  
)
```

A7.2.3.3.2 Example CCA platform claims

I_{TVHKL}

An example CCA platform claim map is shown below in COSE-DIAG format:

```
/ CCA platform claim map /
{
  / cca-platform-profile /
  265: "tag:arm.com,2024:cca_platform#1.1.0",

  / cca-platform-challenge /
  10: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',

  / cca-platform-implementation-id /
  2396: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',

  / cca-platform-instance-id /
  256: h'010BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
      BB',

  / cca-platform-config /
  2401: h'CFCFCFCF',

  / cca-platform-lifecycle /
  2395: 12288,

  / cca-platform-sw-components /
  2399: [
    {
      / measurement value /
      2: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
          AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',

      / signer id /
      5: h'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
          BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB',

      / version /
      4: "1.0.0",

      / hash algorithm identifier /
      6: "sha-256"
    },
    {
      / measurement value /
      2: h'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC',

      / signer id /
      5: h'DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
          DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD',

      / version /
      4: "1.0.0",

      / hash algorithm identifier /
      6: "sha-256"
    }
  ],

  / cca-platform-verification-service /
```

```
2400: "https://cca_verifier.org",  
  
/ cca-platform-hash-algo-id /  
2402: "sha-256"  
}
```

DRAFT

Chapter A8

Realm debug and performance monitoring

This section describes the debug and performance monitoring features which are available to a Realm.

A8.1 Realm PMU

This section describes the programming model for usage of PMU by a Realm.

R_{DNNQO}

On REC entry, Realm PMU state is restored from the REC object.

R_{LHRYJ}

On REC exit, all Realm PMU state is saved to the REC object.

R_{WXTZF}

On REC exit, `exit.pmu_ovf_status` indicates the status of the PMU overflow at the time of the Realm exit.

See also:

- [A3.6 Realm support for Performance Monitors Extension](#)
- [A4.3 REC exit](#)
- [B4.4.33 RmiRecExit type](#)

DRAFT

Chapter A9

Realm device assignment

This section describes how devices are assigned to Realms, attested and granted permission to access Realm-owned memory.

A9.1 Realm device assignment overview

- I₀₁₀₇ The RMM allows a device to be assigned to a Realm in a trustworthy manner, allowing the Realm to attest the identity and configuration of the device before it is permitted to access the Realm's memory.
- I₀₁₀₈ From the Host point of view, devices are managed using the following RMM objects:
- *Physical Device* (PDEV)
Represents a communication channel between the RMM and a physical device, for example a PCIe device.
 - *Virtual Device* (VDEV)
Represents the binding between a device function and a Realm. For example, a VDEV can represent a physical function of a PCIe device or a virtual function of a multi-function PCIe device. Every VDEV is associated with one PDEV.
 - *Virtual SMMU* (VSMMU)
Stores the state of an Arm Virtual System Memory Management Unit (VSMMU) which is emulated by the RMM, allowing the Realm to apply stage 1 translation to transactions initiated by assigned device functions.
- Interaction between the Host and a PDEV, VDEV and VSMMU objects is performed via RMI commands.
- I₀₁₀₉ From the Realm point of view, an assigned device function is represented by a *Realm Device* (RDEV).
Interaction between the Realm and an RDEV is performed via RSI commands.
- U₀₁₁₀ Arm expects that a VDEV and an RDEV will both refer to the same data structure inside the RMM.

A9.1.1 Device protection types

- I₀₁₁₁ Communication between a device and either the RMM or a Realm executing the PE must be protected against attack from software outside the TCB of the RMM or Realm.
- I₀₁₁₂ Communication with a device which is physically integrated into the platform is said to be protected by system construction. For such devices, either SPDM, IDE or both may not be required, depending upon details of the device integration.
- I₀₁₁₃ Communication with a device which is physically separate from the platform must be protected using SPDM and IDE.

A9.1.2 Device attestation types

- D₀₁₁₄ A *platform-attested device* is a device whose identity and firmware are attested as part of the CCA platform.
- I₀₁₁₅ A device which is physically integrated into the platform and which does not implement its own Device Security Manager (DSM) is a platform-attested device.
- D₀₁₁₆ An *independently-attested device* is a device whose identity and firmware are attested separately from the CCA platform.
- I₀₁₁₇ An independently-attested device implements its own DSM.
See also:
- [A3.10 Support for Realm device assignment](#)
 - [A9.3 Physical device object](#)
 - [A9.4 Virtual device object](#)
 - [A9.6 Virtual SMMU](#)

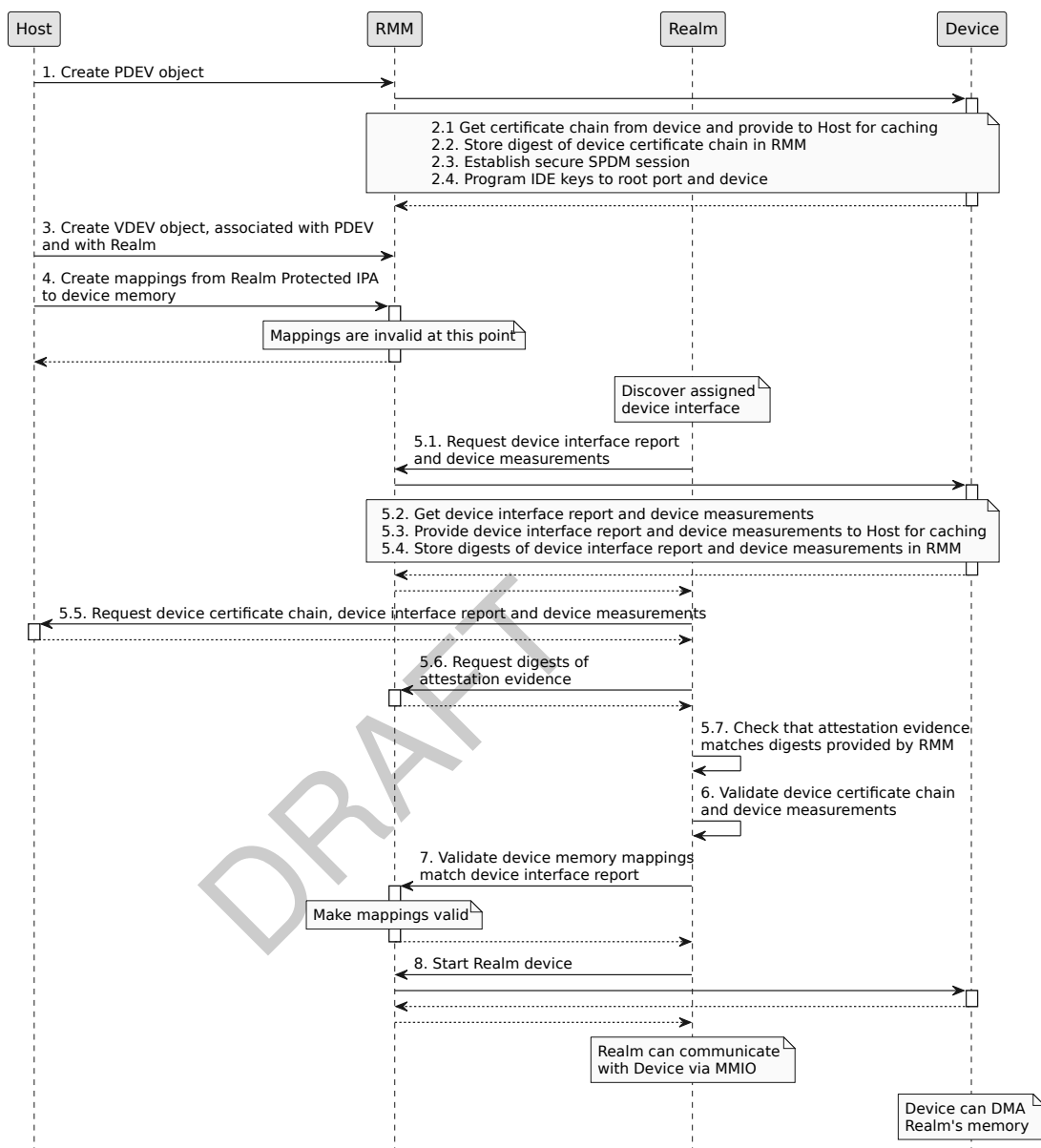
A9.1.2.1 Assignment of an independently-attested device

- I₀₁₁₈ The communication channel between the RMM and an independently-attested device is managed by the Host.

- I₀₁₁₉ Communication between the RMM and an independently-attested device is protected using the Security Protocol and Data Model (SPDM) protocol.
- I₀₁₂₀ Identity of an independently-attested device is described by a device certificate chain.
- I₀₁₂₁ Firmware measurements for an independently-attested device are reported in an SPDM measurement block.
- I₀₁₂₂ Assignment of an independently-attested device to a Realm involves the following steps:
1. The Host creates a PDEV object, associated with the target physical device.
 2. The Host initializes the PDEV object, causing the following to happen:
 - A secure channel is established between the RMM and the device.
 - The physical link between the device and memory is secured. For example, for an off-chip PCIe device, this is achieved using the Integrity and Data Encryption (IDE) standard. See [PCI Express 6.0 specification](#) [14].
 - The device certificate chain is provided to the Host. The Host is expected to store this information, and to later present it to the Realm.
 - A digest of the device certificate chain is stored by the RMM. This is used later to check integrity of the attestation evidence provided by the Host to the Realm.
 3. The Host creates a VDEV object, which represents a binding between a function of the target device, and a Realm. At this stage, the target device is not granted access to the Realm-owned memory.
 4. The Host maps memory regions of the target device function into the Protected IPA space of the Realm. At this stage, the mappings are invalid, so the Realm cannot yet access the device's memory regions.
 5. Device information provided by the RMM tells the Realm that the device is independently-attested. The Realm requests the RMM to retrieve device attestation evidence (device interface report and device measurements.) As with the device certificate chain, the evidence is provided to the Host for caching, and the RMM stores digests of the evidence. The Realm later requests device attestation evidence from the Host, and verifies that this matches the corresponding digests stored by the RMM.
 6. The Realm verifies that the device identity (represented by the device certificate chain) and device measurements are acceptable.
 7. The Realm verifies that device memory mappings created by the Host match those described in the device interface report. Once each mapping has been verified, it is made valid by the RMM, so the Realm can access the device's memory regions.
 8. The Realm instructs the RMM to grant the device access to Realm-owned memory.
- X₀₁₂₃ Requiring the Host to store device attestation evidence means that storage for this information, whose size may not be known ahead of time, does not need to be allocated in RMM memory. The RMM therefore only needs to store a digest of the Host-cached data, which can be used by the Realm to check integrity of data retrieved from the Host.

I₀₁₂₄

Assignment of an independently-attested device to a Realm is illustrated in the following sequence diagram.



Note that “secure SPDM” means that the requester (the RMM) has verified that the responder holds the private key which was used to sign the device certificate chain. The identity and trustworthiness of the responder are not evaluated until the Realm receives attestation evidence for the device.

R₀₁₂₅

The device certificate chain digest computed by the RMM for an independently-attested device is computed over the concatenation of individual X.509 certificates in the chain, without SPDM header and root certificate hash. The device certificate chain must meet SPDM requirements. The leaf certificate must meet CMA-SPDM requirements.

See also:

- [Secured Messages using SPDM Specification version 1.1.0 \[15\]](#)

A9.1.2.2 Assignment of a platform-attested device

I₀₁₂₆

The communication channel between the RMM and a platform-attested device is IMPLEMENTATION DEFINED.

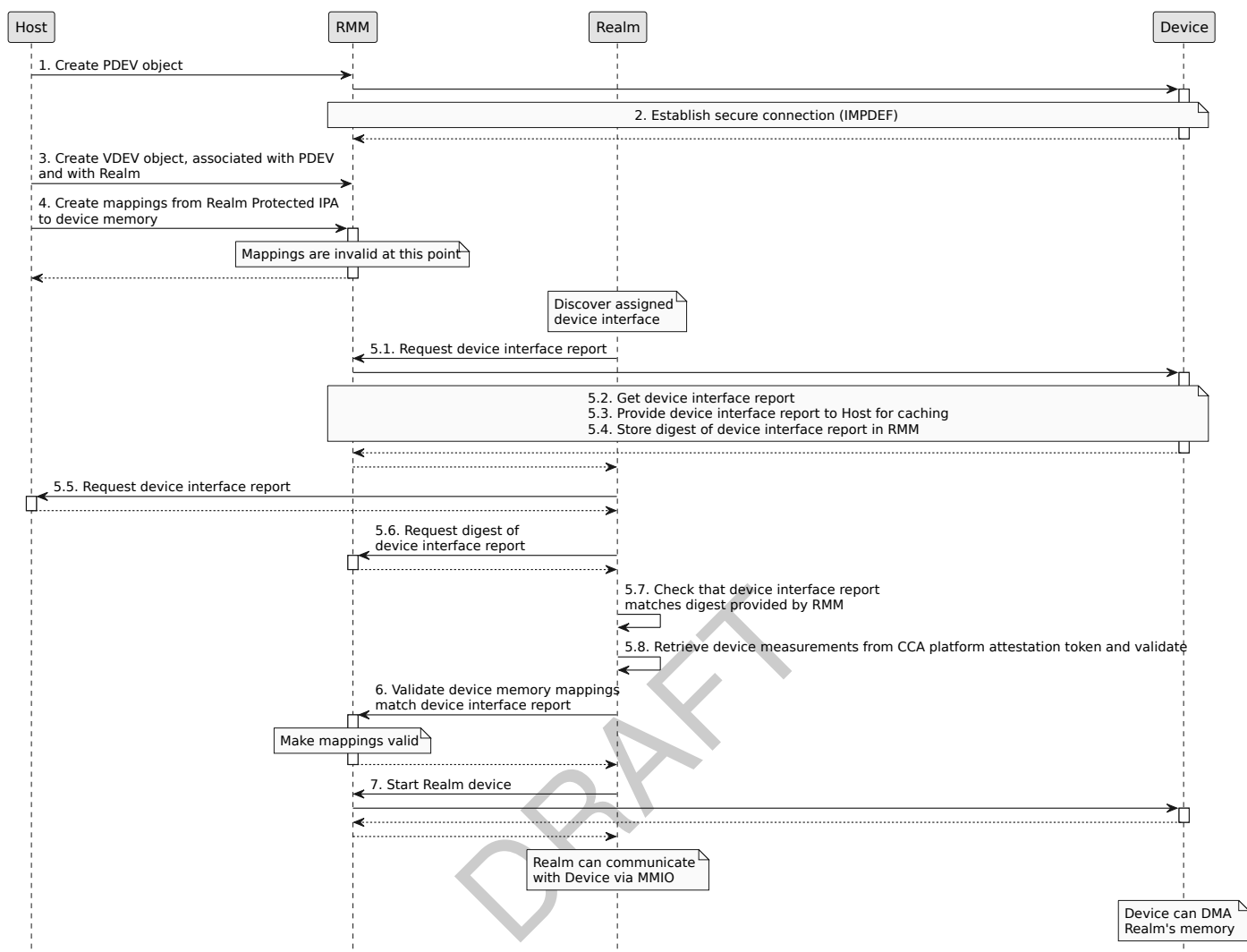
I₀₁₂₇ Firmware measurements for a platform-attested device are reported in the CCA platform software components claim.

I₀₁₂₈ Assignment of a platform-attested device to a Realm involves the following steps:

1. The Host creates a PDEV object, associated with the target physical device.
2. The Host initializes the PDEV object, causing the following to happen:
 - A secure channel is established between the RMM and the device.
3. The Host creates a VDEV object, which represents a binding between a function of the target device, and a Realm. At this stage, the target device is not granted access to the Realm-owned memory.
4. The Host maps memory regions of the target device function into the Protected IPA space of the Realm. At this stage, the mappings are invalid, so the Realm cannot yet access the device's memory regions.
5. Device information provided by the RMM tells the Realm that the device is platform-attested. The Realm requests the RMM to retrieve a device interface report. The device interface report is provided to the Host for caching, and the RMM stores digests of the device interface report. The Realm later requests the device interface report from the Host, and verifies that this matches the corresponding digest stored by the RMM.
6. The Realm retrieves device measurements from the CCA platform attestation token, and verifies that the measurements are acceptable.
7. The Realm verifies that device memory mappings created by the Host match those described in the device interface report. Once each mapping has been verified, it is made valid by the RMM, so the Realm can access the device's memory regions.
8. The Realm instructs the RMM to grant the device access to Realm-owned memory.

I₀₁₂₉ Assignment of a platform-attested device to a Realm is illustrated in the following sequence diagram.

Chapter A9. Realm device assignment
A9.1. Realm device assignment overview



See also:

- [A7.2.3.2.7 CCA platform software components claim](#)
- [B5.3.16 RSI_RDEV_GET_INFO command](#)

A9.2 Communication between RMM and a device

A9.2.1 Device requests and responses

- D₀₁₃₀ Communication between the RMM and a device consists of a series of *device requests* sent from the RMM to the device and *device responses* returned by the device to the RMM.
- D₀₁₃₁ A *device transaction* is a series of one or more (device request, device response) tuples.
- I₀₁₃₂ At the requester side (that is, at the RMM), a device transaction is associated with either a PDEV or a VDEV, depending on the event which triggered the device transaction:
- If the device transaction was triggered by one of the following RMI commands then the device transaction is associated with the PDEV.
 - RMI_PDEV_CREATE
 - RMI_PDEV_SET_PUBKEY
 - RMI_PDEV_NOTIFY
 - RMI_PDEV_STOP
 - If the device transaction was triggered by one of the following RMI commands then the device transaction is associated with the VDEV.
 - RMI_VDEV_STOP
 - If the device transaction was triggered by one of the following RSI commands then the device transaction is associated with the VDEV.
 - RSI_RDEV_GET_INTERFACE_REPORT
 - RSI_RDEV_GET_MEASUREMENTS
 - RSI_RDEV_LOCK
 - RSI_RDEV_START
 - RSI_RDEV_STOP
- I₀₁₃₃ A PDEV is associated with at most one device transaction at a time.
- I₀₁₃₄ A VDEV is associated with at most one device transaction at a time.
- I₀₁₃₅ Communication between the RMM and an off-chip device consists of SPDm messages, transported via Non-secure memory.
- I₀₁₃₆ Communication between the RMM and an on-chip device can occur via one of two paths:
- SPDm messages, transported via Non-secure memory.
 - Platform communication, via an IMPLEMENTATION DEFINED channel.
- I₀₁₃₇ When the RMM requires the Host to send a device request, the “send” flag is set in the DevCommExitFlags fieldset.
- I₀₁₃₈ When the RMM sends a device request via an IMPLEMENTATION DEFINED channel, the “send” flag is clear and the “wait” flag is set in the DevCommExitFlags fieldset. This informs the Host that it should either:
- Register for an IMPLEMENTATION DEFINED notification that the request has been completed, and call RMI_PDEV_COMMUNICATE or RMI_VDEV_COMMUNICATE when this notification is received, or
 - Poll for request completion by periodically calling RMI_PDEV_COMMUNICATE or RMI_VDEV_COMMUNICATE.
- D₀₁₃₉ The states of a Device communication are listed below.

State	Description
DEV_COMM_IDLE	The RMM is not communicating with the device.
DEV_COMM_PENDING	The RMM has a device request which is ready to be sent to the device.

State	Description
DEV_COMM_ACTIVE	The RMM has initiated a device transaction. One or more device requests associated with this device transaction have been sent from the RMM to the device. The RMM has not received all the expected device responses associated with this device transaction.
DEV_COMM_ERROR	The RMM encountered an error during communication with the device.

I₀₁₄₀

Device communication state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

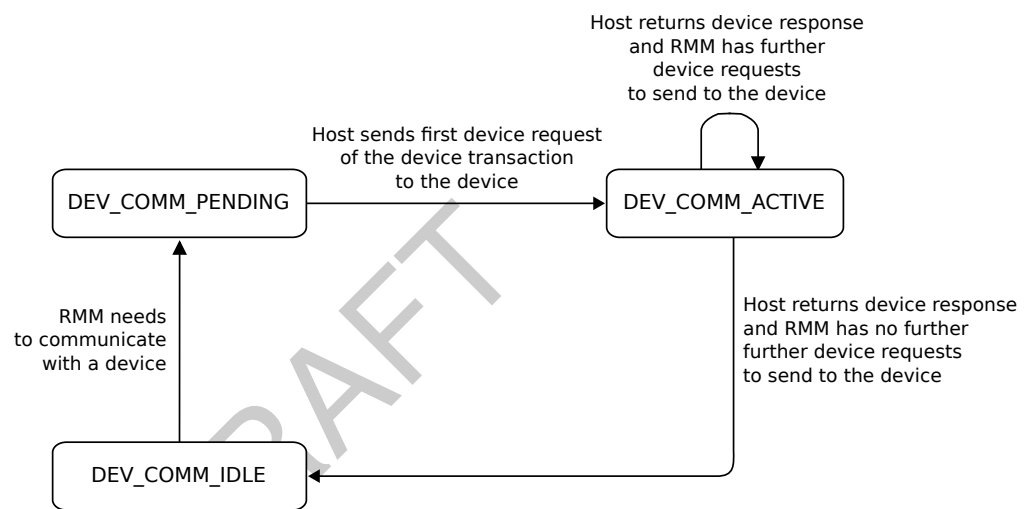


Figure A9.1: Device communication state transitions

See also:

- [Secured Messages using SPDm Specification version 1.1.0 \[15\]](#)
- [A9.5.1 Interruptible Realm device operations](#)
- [B4.4.9 RmiDevCommExitFlags type](#)

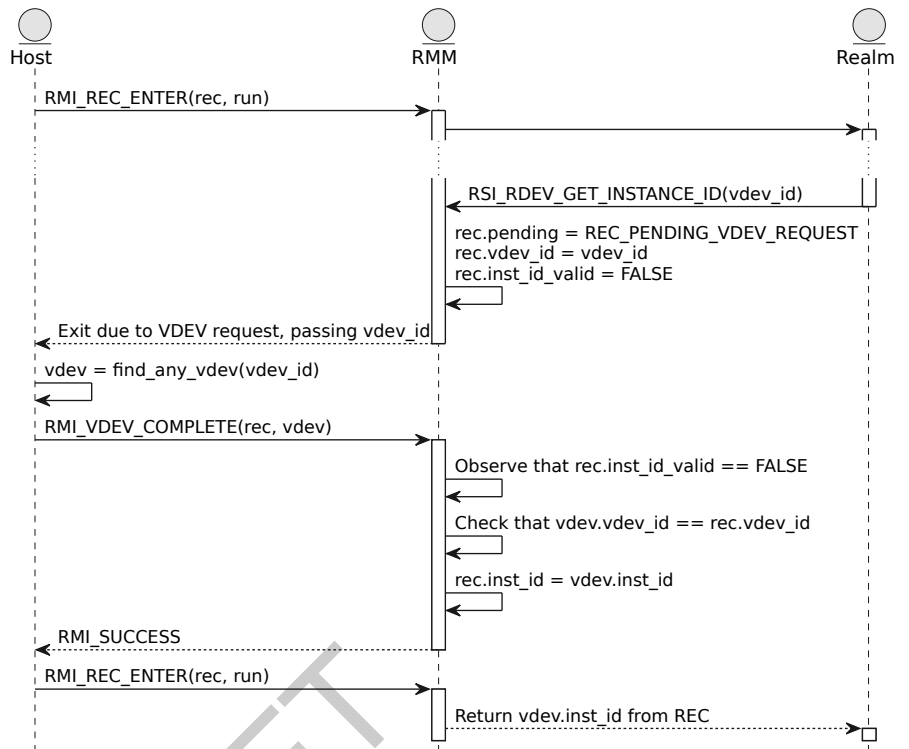
A9.2.2 Mapping from virtual device ID to VDEV object

I₀₁₄₁

The RMM does not maintain a mapping from virtual device ID to VDEV object. Consequently, when a Realm executes an RSI_RDEV command, passing a virtual device ID, the RMM must request the Host to provide a corresponding VDEV object.

I₀₁₄₂

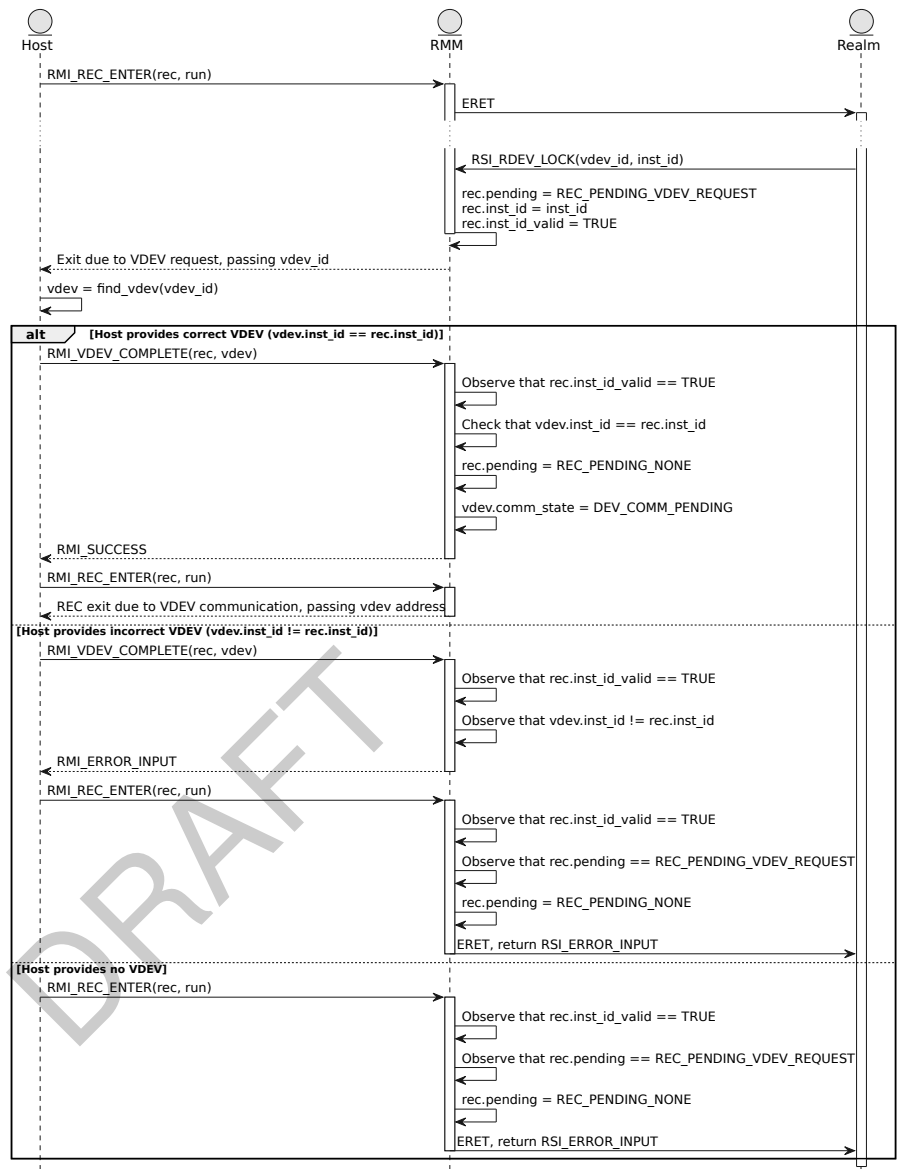
The following sequence diagram shows how the virtual device ID passed by a Realm is mapped to a corresponding VDEV object.



I₀₁₄₃ RSI_RDEV_GET_INSTANCE_ID returns a “device instance ID” value. The device instance ID is guaranteed to be unique among VDEVs owned by the calling Realm, unlike the virtual device ID. When the Realm calls any RSI_RDEV command other than RSI_RDEV_GET_INSTANCE_ID, it passes the (virtual device ID, instance ID) tuple.

I₀₁₄₄ The following sequence diagram shows how the (virtual device ID, instance ID) tuple passed by a Realm is mapped to the corresponding VDEV object.

If the mapping is not completed (because either the Host provided the incorrect VDEV object, or provided no VDEV object), an error is returned to the Realm following the next REC entry.



See also:

- [A4.3.12 REC exit due to VDEV request](#)
- [B4.3.51 RMI_VDEV_COMPLETE command](#)
- [B5.3.15 RSI_RDEV_CONTINUE command](#)
- [B5.3.16 RSI_RDEV_GET_INFO command](#)
- [B5.3.18 RSI_RDEV_GET_INTERFACE_REPORT command](#)
- [B5.3.19 RSI_RDEV_GET_MEASUREMENTS command](#)
- [B5.3.20 RSI_RDEV_GET_STATE command](#)
- [B5.3.21 RSI_RDEV_LOCK command](#)
- [B5.3.22 RSI_RDEV_START command](#)
- [B5.3.23 RSI_RDEV_STOP command](#)
- [B5.3.24 RSI_RDEV_VALIDATE_MAPPING command](#)

A9.2.3 Host-side device communication flow

I₀₁₄₅

The RMI_PDEV_COMMUNICATE command is used to send a PDEV-associated device request from the RMM to a device, and / or to return a device response from the device to the RMM.

- I₀₁₄₆ The RMI_VDEV_COMMUNICATE command is used to send a VDEV-associated device request from the RMM to a device, and / or to return a device response from the device to the RMM.
- I₀₁₄₇ The RMI_PDEV_COMMUNICATE and RMI_VDEV_COMMUNICATE commands have identical programming models. Hereafter, they are referred to collectively as “device communication commands”.
- R₀₁₄₈ For a given physical device, at most one device transaction can be active.
- I₀₁₄₉ The RMI_PDEV_ABORT command or RMI_VDEV_ABORT command is used to abort an DEV_COMM_ACTIVE device transaction.
- I₀₁₅₀ At the responder side (that is, at the device), device transactions associated with a PDEV and device transactions associated with its child VDEVs all terminate at the same physical device.

A9.2.3.1 Communication flow for devices which use SPDM

- I₀₁₅₁ The overall flow for communication between the RMM and a device which uses SPDM is as follows:
1. The RMM indicates to the Host that a device transaction, associated with a specified PDEV or VDEV, is DEV_COMM_PENDING.
 - If the device transaction was triggered by execution of an RMI command, this indication is provided via the command’s output values.
 - If the device transaction was triggered by execution of an RSI command, this indication is provided via a REC exit due to VDEV communication.

The RMM also indicates to the Host whether the pending transaction will contain more than one (device request, device response) tuple.
 2. The Host calls the appropriate device communication command.

Input values to the command include a *device request buffer*, which is a pointer to a Granule of NS memory.
 3. The RMM writes a device request to the device request buffer and returns to the Host, indicating that data is ready to be sent to the device. The device request is guaranteed to be no larger than 4KB.
 4. The Host sends the device request to the device.

Details of how this is performed are out of scope of this specification. For an off-chip PCIe device, this could be done by copying from the device request buffer to a Data Object Exchange (DOE) mailbox.

The device communication state becomes DEV_COMM_ACTIVE.
 5. The device indicates to the Host that it has responded to the request.

The Host copies the device response to a *device response buffer* in NS memory. As with sending the request, details of how this are out of scope of this specification.
 6. The Host calls the device communication command, providing a pointer to the device response buffer.
 7. The return value indicates that either:
 - The RMM has another device request to send within the same device transaction (in which case the flow returns to step 2), or
 - The device transaction is complete.
 8. If the device transaction is incomplete, the Host checks the device state to determine whether an error has occurred.
- I₀₁₅₂ When multiple device transactions destined for the same physical device are DEV_COMM_PENDING, the Host is free to choose which of them to transition to DEV_COMM_ACTIVE.
- I₀₁₅₃ When a device transaction is DEV_COMM_ACTIVE, the Host must not send to that physical device a device request associated with any other device transaction.

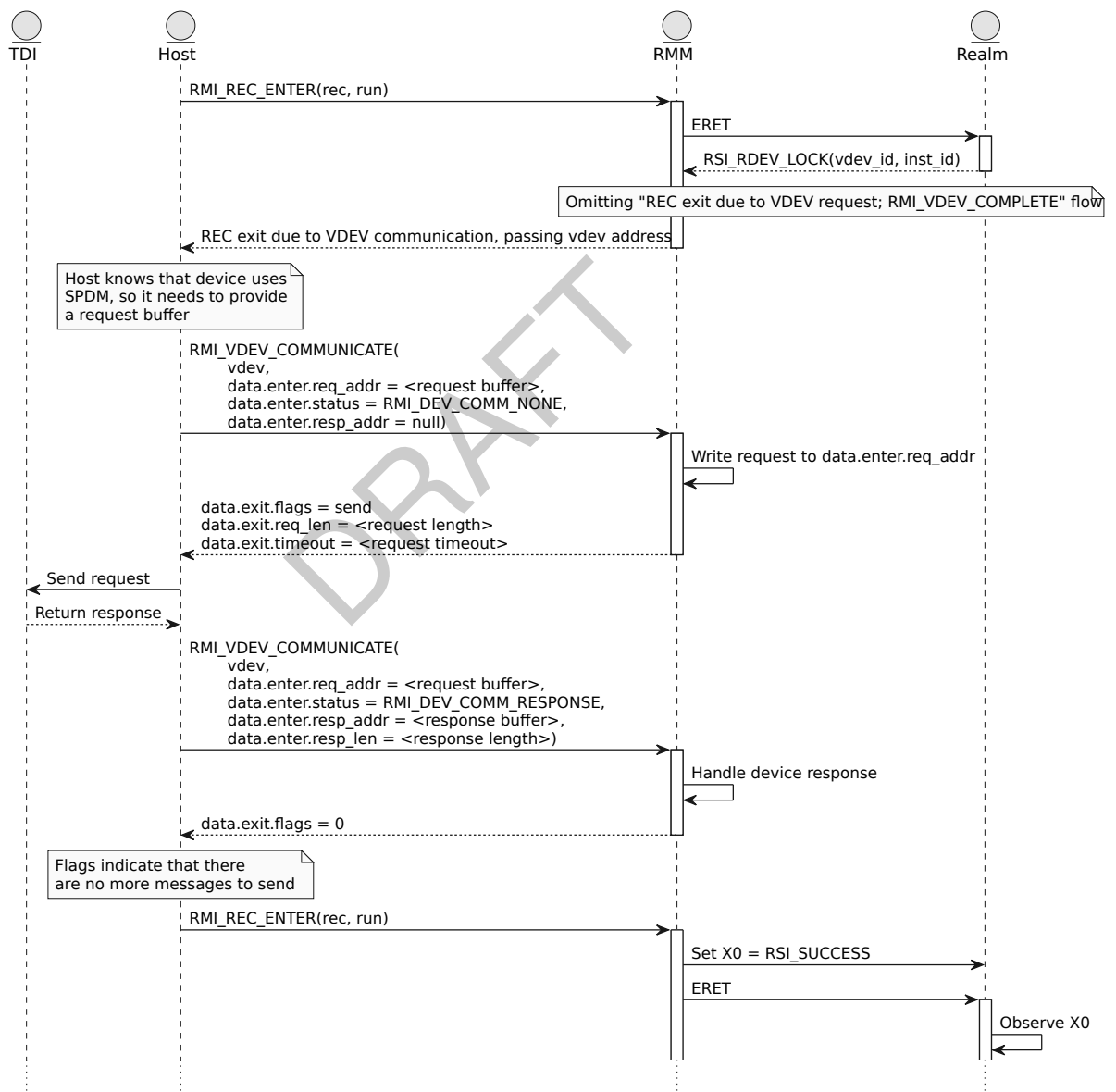
U₀₁₅₄ The SPDM session which is established between the RMM and a device has the following characteristics:

- SPDM measurements use DMTF format
- SPDM heartbeat is not supported
- SPDM key update is not supported

U₀₁₅₅ In order for its functions to be assignable to Realms, a device must provide the following functionality:

- SPDM version required by PCIe TDISP and IDE_KM specifications.
- Identity and authentication including key exchange.

I₀₁₅₆ The following sequence illustrates communication with a device which uses SPDM, taking RSI_RDEV_LOCK as the example which initiates the communication.



See also:

- [PCI Express 6.0 specification \[14\]](#)
- [A4.3.13 REC exit due to VDEV communication](#)

A9.2.3.2 Communication flow for devices which do not use SPDm

I₀₁₅₇

The overall flow for communication between the RMM and a device which does not use SPDm is as follows:

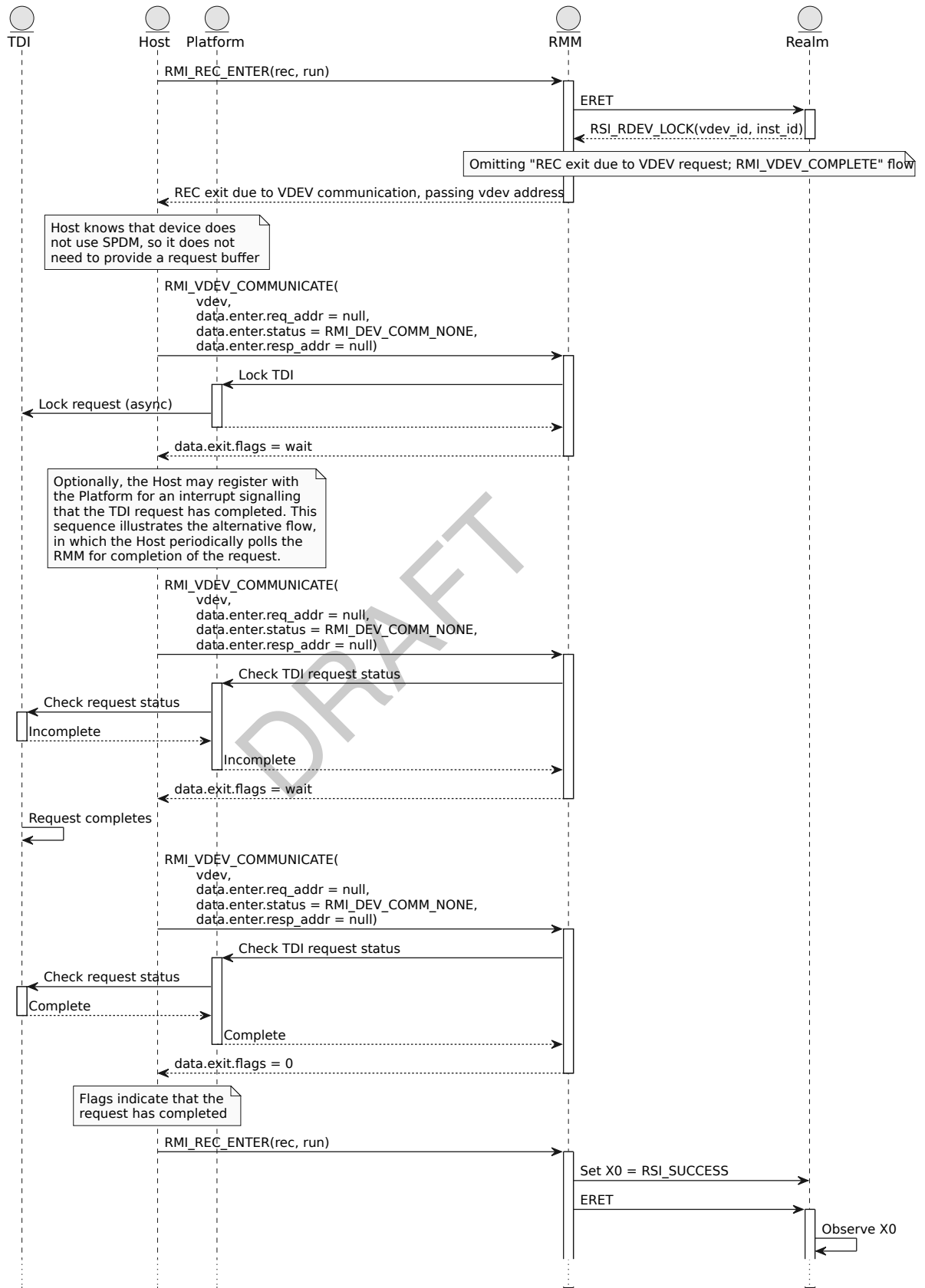
1. The RMM indicates to the Host that a device transaction, associated with a specified PDEV or VDEV, is DEV_COMM_PENDING.
 - If the device transaction was triggered by execution of an RMI command, this indication is provided via the command's output values.
 - If the device transaction was triggered by execution of an RSI command, this indication is provided via a REC exit due to VDEV communication.
2. The Host calls the appropriate device communication command repeatedly, until either:
 - The return value indicates that the device transaction is complete.
 - The device enters an error state.

I₀₁₅₈

The following sequence illustrates communication with a device which does not use SPDm, taking RSI_RDEV_LOCK as the example which initiates the communication.

DRAFT

Chapter A9. Realm device assignment
A9.2. Communication between RMM and a device



See also:

- [A4.3.13 REC exit due to VDEV communication](#)

A9.2.4 Host caching of device attestation evidence

I₀₁₅₉ On execution of a device communication command, the RMM can indicate to the Host that the Host should cache data from the response buffer, for later retrieval by the Realm.

The identity of the data which the Host is requested to cache is implied by the command which triggered the device transaction, as follows:

- If the device transaction was triggered while the PDEV state was PDEV_NEW then the cached data is a device certificate chain.
- If the device transaction was triggered by RSI_RDEV_GET_INTERFACE_REPORT then the cached data is a device interface report.
- If the device transaction was triggered by RSI_RDEV_GET_MEASUREMENTS then the cached data is a device measurement block.

See also:

- [A4.3.13 REC exit due to VDEV communication](#)
- [A9.5.2 Realm retrieval of device attestation evidence](#)
- [B4.3.14 RMI_PDEV_CREATE command](#)
- [B5.3.18 RSI_RDEV_GET_INTERFACE_REPORT command](#)
- [B5.3.19 RSI_RDEV_GET_MEASUREMENTS command](#)

A9.2.5 Device communication data structures

A9.2.5.1 Device communication exit data structure

D₀₁₆₀ An *RmiDevCommExit* object is a data structure which is passed from the RMM to the Host during a device transaction.

I₀₁₆₁ The attributes of an *RmiDevCommExit* object tell the Host the following:

- Whether there is data in the device response buffer which the Host is requested to cache. This is indicated by the *RmiDevCommExitFlags::cache* flag.
- Whether the device request buffer contains a device request which the Host is requested to send to the device. This is indicated by the *RmiDevCommExitFlags::send* flag.
- Whether the RMM is waiting for a response from the device. This is indicated by the *RmiDevCommExitFlags::wait* flag.
- Whether the device transaction contains more than one (device request, device response) tuple. This is indicated by the *RmiDevCommExitFlags::multi* flag.

I₀₁₆₂ *RmiDevCommExitFlags::send* and *RmiDevCommExitFlags::wait* are never set together.

I₀₁₆₃ *RmiDevCommExitFlags::multi* is only set when *RmiDevCommExitFlags::send* is set.

I₀₁₆₄ During communication between the RMM and a device which uses SPDm:

- *RmiDevCommExitFlags::send* indicates that the Host is requested to send a device request to the device.
- *RmiDevCommExitFlags::wait* indicates that the Host is requested to return a device response to the RMM.

I₀₁₆₅ During communication between the RMM and a device which does not use SPDm:

- *RmiDevCommExitFlags::wait* indicates that the Host is requested to notify the RMM when a device response is available.

D₀₁₆₆ The attributes of an *RmiDevCommExit* object are summarized in the following table.

Name	Byte offset	Type	Description
flags	0x0	RmiDevCommExitFlags	Flags indicating action(s) which the Host is requested to perform
cache_offset	0x8	UInt64	If flags.cache is true, offset in the device response buffer to the start of data to be cached, in bytes
cache_len	0x10	UInt64	If flags.cache is true, amount of data to be cached, in bytes
protocol	0x18	RmiDevCommProtocol	If flags.send is true, protocol to use
req_len	0x20	UInt64	If flags.send is true, amount of valid data in request buffer in bytes
timeout	0x28	UInt64	If flags.wait is true, amount of time to wait for device response in milliseconds

See also:

- [B4.4.8 RmiDevCommExit type](#)
- [B4.4.9 RmiDevCommExitFlags type](#)

A9.2.5.2 Device communication enter data structure

D₀₁₆₇ An *RmiDevCommEnter* object is a data structure which is passed from the Host to the RMM during a device transaction.

I₀₁₆₈ The attributes of an *RmiDevCommEnter* object tell the RMM the following:

- Whether the device reported an error.
- Whether the device response buffer contains a device response.
- The location of the device request buffer, into which the RMM can write a device request.

D₀₁₆₉ The attributes of an *RmiDevCommEnter* object are summarized in the following table.

Name	Byte offset	Type	Description
status	0x0	RmiDevCommStatus	Status of device transaction
req_addr	0x8	Address	Address of request buffer
resp_addr	0x10	Address	Address of response buffer
resp_len	0x18	UInt64	Amount of valid data in response buffer in bytes

See also:

- [B4.4.7 RmiDevCommEnter type](#)

A9.3 Physical device object

D₀₁₇₀ A *Physical Device* (PDEV) represents a communication channel between the RMM and a physical device, for example a PCIe device.

A9.3.1 Physical device attributes

D₀₁₇₁ The attributes of a PDEV are summarized in the following table.

Name	Type	Description
pdev_id	Bits64	Device identifier
prot	RmmPdevProtection	Configuration of protection between system and device
segment_id	Bits8	Segment identifier PCIe Segment identifier of the Root Port and endpoint.
ecam_addr	Address	ECAM base address of the PCIe configuration space.
root_id	Bits16	Root Port identifier Physical PCIe routing identifier of the Root Port to which the endpoint is connected.
cert_id	UInt64	Certificate identifier
rid_base	Bits16	Base of requester ID range (inclusive). The value is in PCI BDF format.
rid_top	Bits16	Top of requester ID range (exclusive). The value is in PCI BDF format.
hash_algo	RmmHashAlgorithm	Algorithm used to generate device digests
ide_sid	UInt64	IDE stream ID
ncoh_num_addr_range	UInt64	Number of device non-coherent address ranges
ncoh_addr_range	RmmAddressRange[16]	Device non-coherent address range
coh_num_addr_range	UInt64	Number of device coherent address ranges
coh_addr_range	RmmAddressRange[4]	Device coherent address range
aux	Address[32]	Addresses of auxiliary Granules
num_aux	UInt64	Number of auxiliary Granules
state	RmmPdevState	Lifecycle state
comm_state	RmmDevCommState	Device communication state
num_vdevs	UInt64	Number of VDEVs associated with this PDEV

I₀₁₇₂ pdev.root_id and pdev.cert_id attributes are ignored for platform-attested devices.

R₀₁₇₃ If pdev.prot is PDEV_IQCOH_E2E_IDE or PDEV_FCOH_E2E_IDE then all of the following are true:

- The RMM checks that pdev.ide_sid is not used by any other PDEV which is connected to the same Root Port.
- The RMM configures an IDE selective stream with IDE selective stream ID pdev.ide_sid and IDE selective stream address ranges pdev.ncoh_addr_range.

R₀₁₇₄ The RMM checks that all entries in (pdev.rid_base, pdev.rid_top] are not used by any other PDEV within the same PCIe segment.

- R₀₁₇₅ The RMM checks that all entries in pdev.ncoh_addr_range have the following properties:
- Fall within non-coherent IO ranges of the system address map.
 - Not used by any other PDEV.
- R₀₁₇₆ If pdev.prot is PDEV_FCOH_E2E_IDE or PDEV_FCOH_E2E_SYS then the RMM checks that all entries in pdev.coh_addr_range have the following properties:
- Map to device coherent memory.
 - Not used by any other PDEV.
- I₀₁₇₇ The following table summarises which PDEV attributes are valid for each value of pdev.prot.

prot	Device type	ide_sid	(rid_base, rid_top]	ncoh_addr_range	coh_addr_range
PDEV_IOCOH_E2E_IDE	PCIe off-chip	Y	Y	Y	
PDEV_IOCOH_E2E_SYS	PCIe on-chip		Y	Y	
PDEV_FCOH_E2E_IDE	CHI off-chip	Y	Y	Y	Y
PDEV_FCOH_E2E_SYS	CHI on-chip		Y	Y	Y

See also:

- [A2.2.1 Granule category](#)

A9.3.2 Physical device lifecycle

A9.3.2.1 States

- D₀₁₇₈ The states of a PDEV are listed below.

State	Description
PDEV_NEW	Initial state of the device.
PDEV_NEEDS_KEY	RMM needs device public key.
PDEV_HAS_KEY	RMM has device public key.
PDEV_READY	Secure connection between the RMM and the device has been established. Physical link between the device and memory is secured. Ready for creation of VDEV instances.
PDEV_IDE_RESETTING	The PDEV's IDE link is being reset.
PDEV_COMMUNICATING	The RMM is communicating with the device.
PDEV_STOPPING	The RMM is communicating with the device to terminate the secure connection between the RMM and the device.
PDEV_STOPPED	Secure connection between the RMM and the device has been terminated.
PDEV_ERROR	Device has reported a fatal error.

A9.3.2.2 State transitions

I₀₁₇₉

Permitted PDEV Realm state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of a PDEV. A transition to the pseudo-state *NULL* represents destruction of a PDEV.

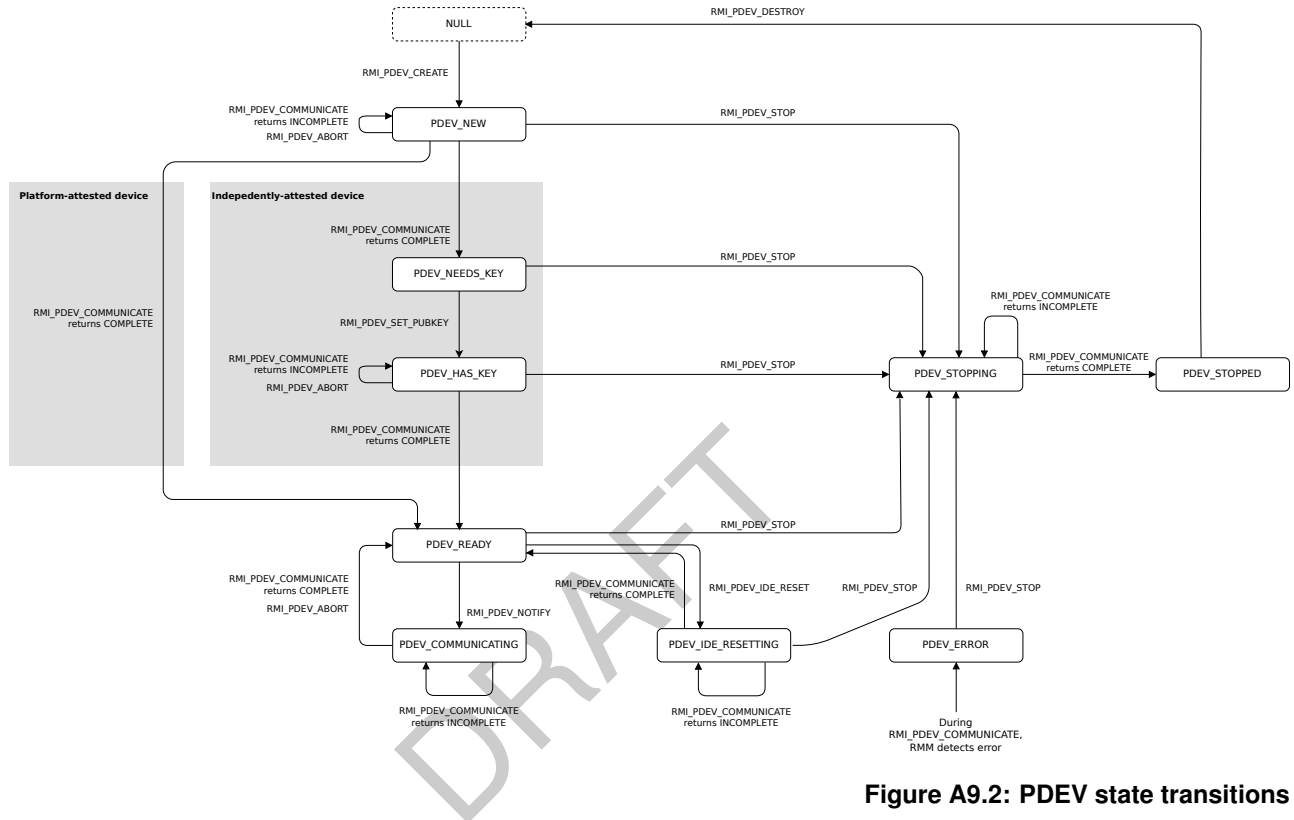


Figure A9.2: PDEV state transitions

A9.3.2.2.1 State transitions for an independently-attested device

I₀₁₈₀

While a PDEV associated with an independently-attested device is in PDEV_NEW state, execution of RMI_PDEV_COMMUNICATE causes the RMM to fetch the device certificate chain. The RMM stores a digest of the device certificate chain for later retrieval by the Realm. The Host is expected to cache the device certificate chain for later retrieval by the Realm.

Once device certificate chain retrieval is complete, the PDEV moves to PDEV_NEEDS_KEY state.

I₀₁₈₁

While a PDEV associated with an independently-attested device is in PDEV_HAS_KEY state, execution of RMI_PDEV_COMMUNICATE causes the RMM to perform secure SPDMM session establishment and IDE key programming.

Once secure SPDM session establishment and IDE key programming is complete, the PDEV moves to PDEV_READY state.

A9.3.2.2.2 State transitions for a platform-attested device

I0182

While a PDEV associated with an platform-attested device is in PDEV_NEW state, execution of RMI PDEV COMMUNICATE causes the RMM to establish a communication channel with the device.

Once device certificate chain retrieval is complete, the PDEV moves to PDEV_READY state.

A9.3.2.2.3 State transitions for devices which use IDE

I₀₁₈₃ While a PDEV is in PDEV_READY state, the Host can notify the RMM of an event relevant to the device by executing RMI_PDEV_NOTIFY.

If the return value is RMI_SUCCESS then the PDEV moves to PDEV_COMMUNICATING state.

I₀₁₈₄ While a PDEV is in PDEV_READY state, if the device class is PCIe then the Host can request the device's IDE link to be reset by executing RMI_PDEV_IDE_RESET.

The PDEV moves to PDEV_IDE_RESETTING state.

I₀₁₈₅ While a PDEV is in PDEV_IDE_RESETTING state, the Host can enact the "IDE reset" device transaction by executing RMI_PDEV_COMMUNICATE.

If the return value indicates that the device transaction is complete then the PDEV moves to PDEV_READY state.

A9.3.2.2.4 State transitions for all devices

I₀₁₈₆ While a PDEV is in PDEV_COMMUNICATING state, the Host can either:

- Transfer device requests and device responses by executing RMI_PDEV_COMMUNICATE. If the return value indicates that the device transaction is complete then the PDEV moves to PDEV_READY state.
- Abort the device transaction by executing RMI_PDEV_ABORT. On successful execution of this command, the PDEV moves to PDEV_READY state.

I₀₁₈₇ On execution of RMI_PDEV_COMMUNICATE, if the RMM detects a fatal error (such as an unexpected response or a protocol error) then the PDEV moves to PDEV_ERROR state.

I₀₁₈₈ While a PDEV is in any of the following state, the Host can request the RMM to stop the device by executing RMI_PDEV_STOP:

- PDEV_NEW
- PDEV_NEEDS_KEY
- PDEV_HAS_KEY
- PDEV_READY
- PDEV_IDE_RESETTING
- PDEV_ERROR

The PDEV moves to PDEV_STOPPING state.

I₀₁₈₉ While a PDEV is in PDEV_STOPPING state, the Host can enact the "stop" device transaction by executing RMI_PDEV_COMMUNICATE.

If the return value indicates that the device transaction is complete then the PDEV moves to PDEV_STOPPED state.

I₀₁₉₀ While a PDEV is in PDEV_STOPPING state, if the device fails to respond or reports an error then the Host can call RMI_PDEV_COMMUNICATE, passing RMI_DEV_COMM_ERROR.

On successful execution of this command, the PDEV moves to PDEV_STOPPED state.

I₀₁₉₁ While a PDEV is in PDEV_STOPPED state, the Host can reclaim resources by executing RMI_PDEV_DESTROY.

This command will fail if the PDEV is associated with any VDEVs.

I₀₁₉₂ Permitted PDEV state transitions are shown in the following table. The rightmost column lists the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of a PDEV object. A transition to the pseudo-state *NULL* represents destruction of a PDEV object.

From state	To state	Events
<i>NULL</i>	PDEV_NEW	RMI_PDEV_CREATE
PDEV_NEW	<i>NULL</i>	RMI_PDEV_DESTROY
PDEV_NEW	PDEV_NEW	RMI_PDEV_ABORT RMI_PDEV_COMMUNICATE
PDEV_NEW	PDEV_NEEDS_KEY	RMI_PDEV_COMMUNICATE
PDEV_NEEDS_KEY	PDEV_HAS_KEY	RMI_PDEV_SET_PUBKEY
PDEV_HAS_KEY	PDEV_HAS_KEY	RMI_PDEV_ABORT RMI_PDEV_COMMUNICATE
PDEV_HAS_KEY	PDEV_READY	RMI_PDEV_COMMUNICATE
PDEV_READY	PDEV_COMMUNICATING	RMI_PDEV_NOTIFY
PDEV_COMMUNICATING	PDEV_COMMUNICATING	RMI_PDEV_COMMUNICATE
PDEV_COMMUNICATING	PDEV_READY	RMI_PDEV_ABORT RMI_PDEV_COMMUNICATE
PDEV_READY	PDEV_IDE_RESETTING	RMI_PDEV_IDE_RESET
PDEV_IDE_RESETTING	PDEV_READY	RMI_PDEV_COMMUNICATE
PDEV_NEW	PDEV_STOPPING	RMI_PDEV_STOP
PDEV_NEEDS_KEY	PDEV_STOPPING	RMI_PDEV_STOP
PDEV_HAS_KEY	PDEV_STOPPING	RMI_PDEV_STOP
PDEV_READY	PDEV_STOPPING	RMI_PDEV_STOP
PDEV_IDE_RESETTING	PDEV_STOPPING	RMI_PDEV_STOP
PDEV_ERROR	PDEV_STOPPING	RMI_PDEV_STOP
PDEV_STOPPING	PDEV_STOPPED	RMI_PDEV_COMMUNICATE
PDEV_STOPPED	<i>NULL</i>	RMI_PDEV_DESTROY

See also:

- [B4.3.11 RMI_PDEV_ABORT command](#)
- [B4.3.13 RMI_PDEV_COMMUNICATE command](#)
- [B4.3.14 RMI_PDEV_CREATE command](#)
- [B4.3.15 RMI_PDEV_DESTROY command](#)
- [B4.3.17 RMI_PDEV_IDE_RESET command](#)
- [B4.3.18 RMI_PDEV_NOTIFY command](#)
- [B4.3.19 RMI_PDEV_SET_PUBKEY command](#)
- [B4.3.20 RMI_PDEV_STOP command](#)

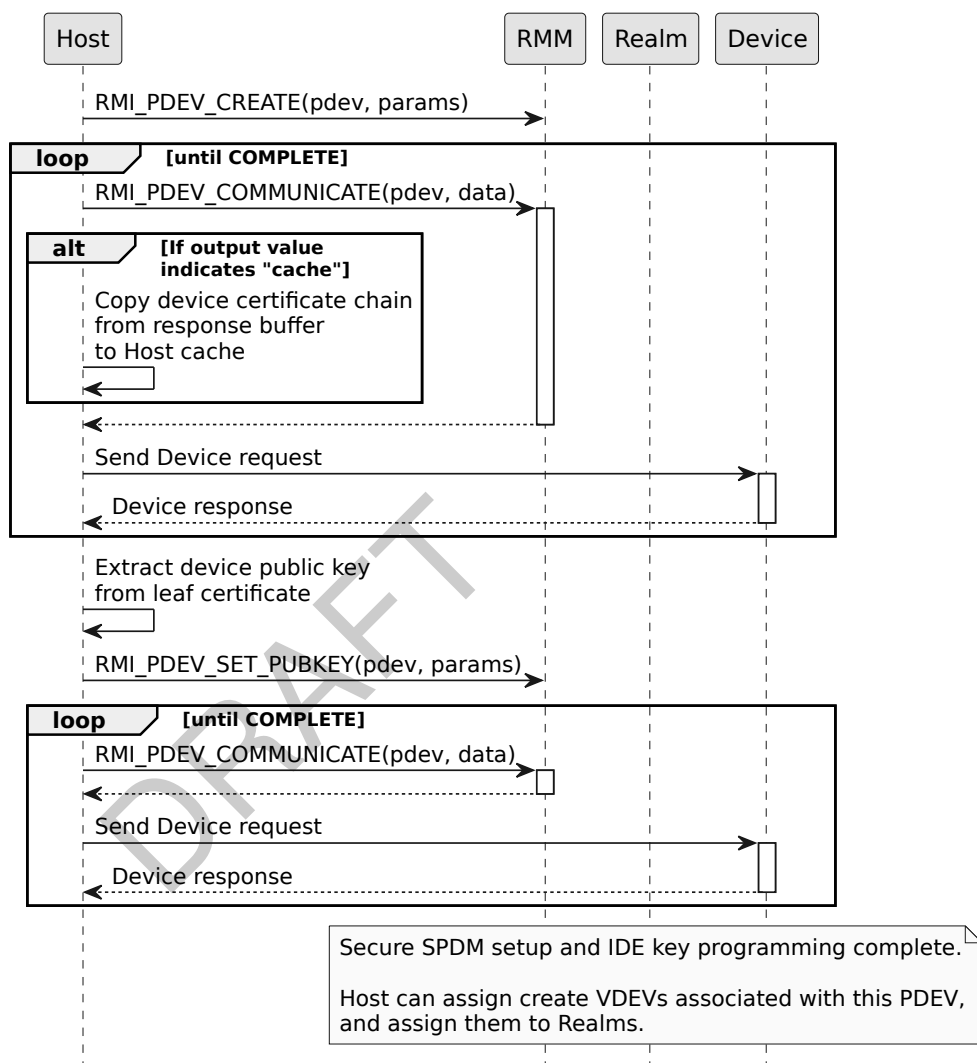
See also:

- [PCI Express 6.0 specification \[14\]](#)
- [A9.2 Communication between RMM and a device](#)

A9.3.3 Physical device flows

A9.3.3.1 Physical device setup flow

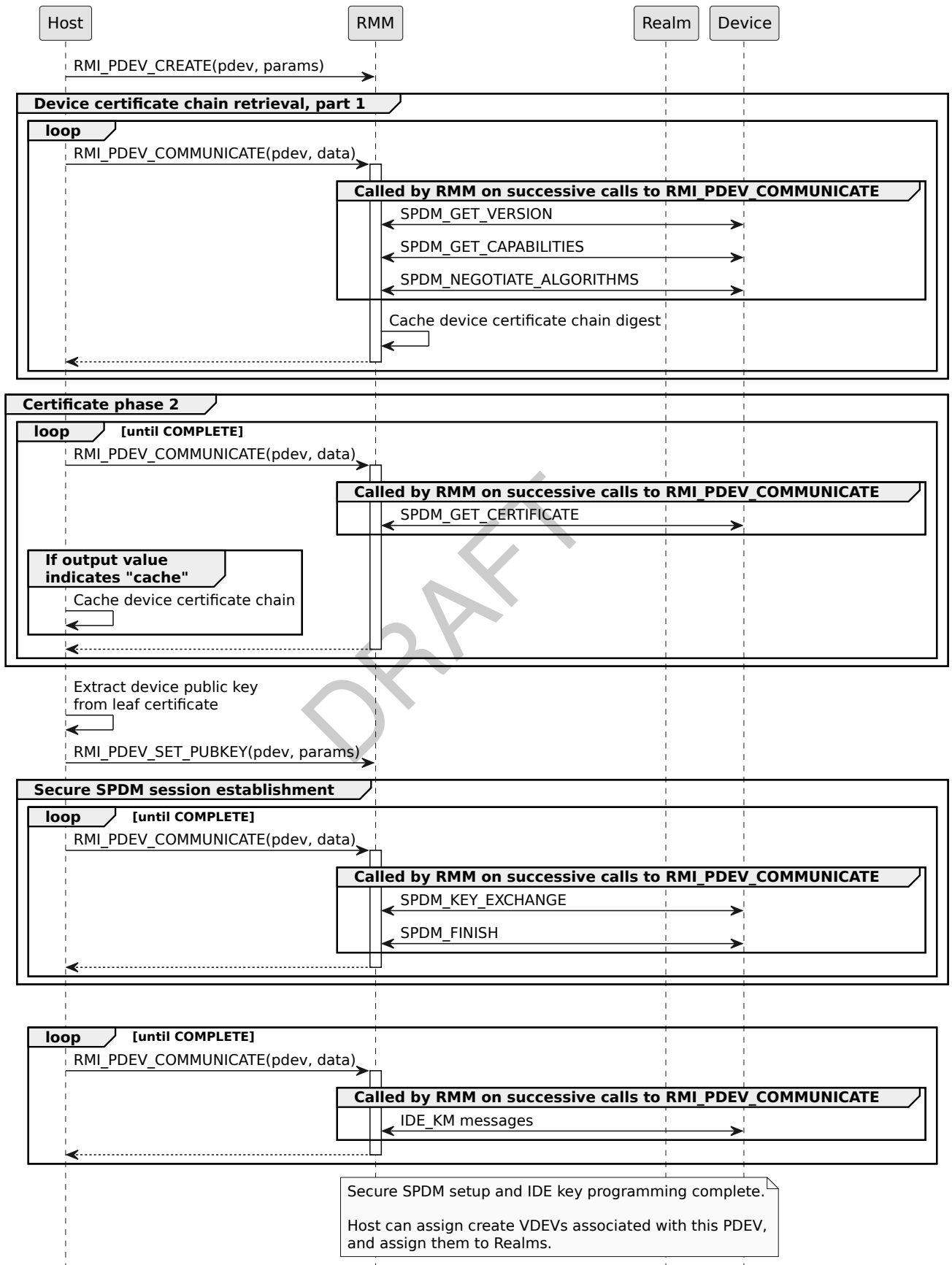
I₀₁₉₃ Setup of a PDEV is illustrated in the following sequence diagram.



I₀₁₉₄

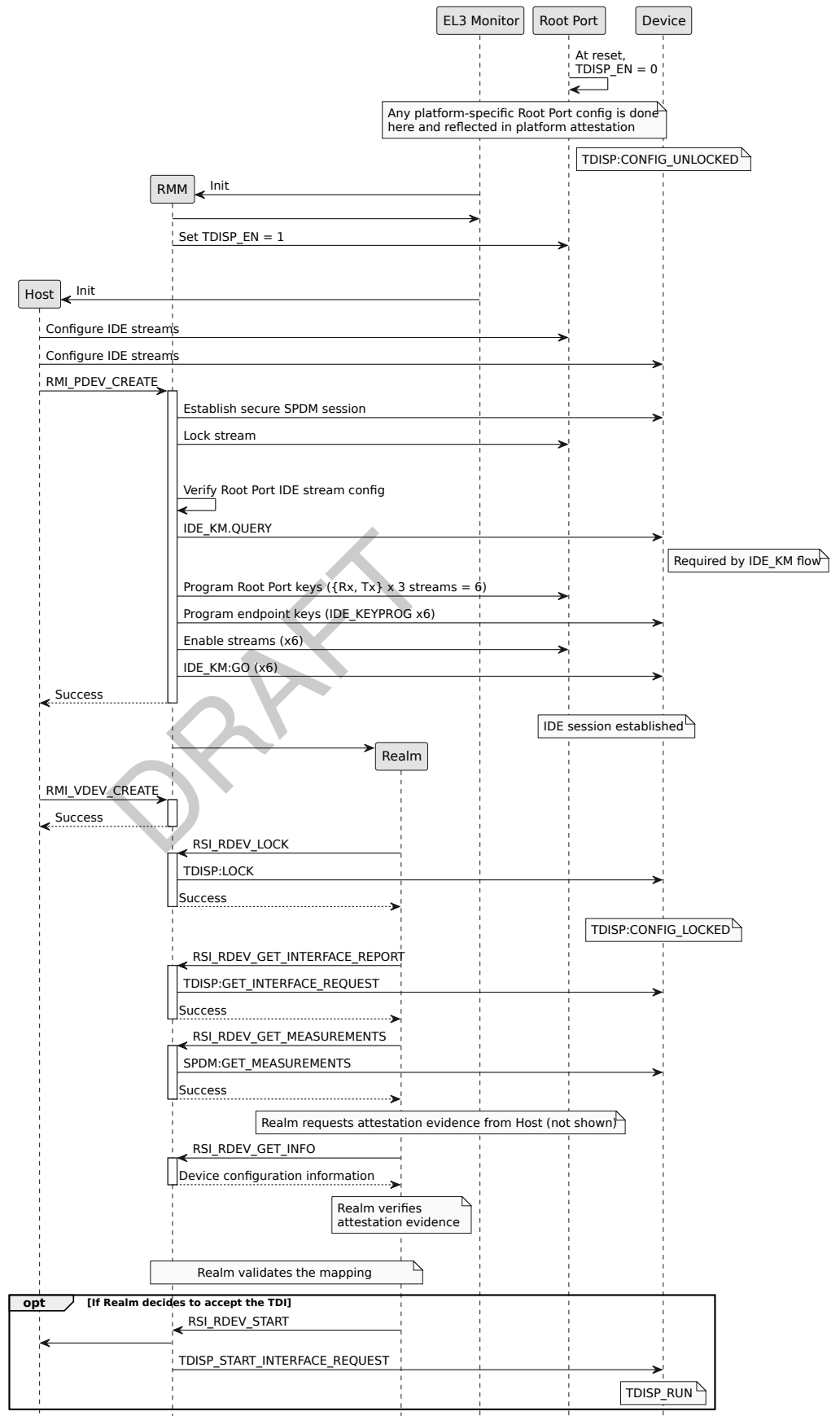
Mapping of the PDEV setup flow onto SPDm and IDE communication with a TDISP PCIe device is illustrated in the following sequence diagram.

DRAFT



I₀₁₉₅ IDE setup for a TDISP PCIe device is illustrated in the following sequence diagram.

DRAFT



See also:

- [PCI Express 6.0 specification \[14\]](#)

DRAFT

A9.4 Virtual device object

D₀₁₉₆ A *virtual device* (VDEV) represents the binding between a device function and a Realm.

A9.4.1 Virtual device attributes

D₀₁₉₇ The attributes of a VDEV are summarized in the following table.

Name	Type	Description
vdev_id	Bits64	Virtual device identifier
tdi_id	Bits64	TDI identifier
inst_id	UInt64	Instance identifier
pdev	Address	PA of parent PDEV
realm	Address	PA of RD of Realm which owns this VDEV
state	RmmVdevState	Lifecycle state
comm_state	RmmDevCommState	Device communication state
aux	Address[32]	Addresses of auxiliary Granules
num_aux	UInt64	Number of auxiliary Granules
vsmmu	RmmFeature	Whether device uses a VSMMU
vsmmu_addr	Address	PA of VSMMU. This field is valid if vsmmu is FEATURE_TRUE.
vsid	Bits64	Virtual Stream Identifier. This field is valid if vsmmu is FEATURE_TRUE.

D₀₁₉₈ A *Device identifier* is a value which identifies a VDEV within a Realm, and is agreed between the Host and the Realm to which the VDEV is assigned.

I₀₁₉₉ The choice of device identifier depends upon the interface provided by the Host to the Realm. For example, if the Host provides a PCIe interface, then identifier is a PCIe (bus, device, function) tuple.

I₀₂₀₀ The Host provides the device identifier when executing RMI_VDEV_CREATE.

I₀₂₀₁ The Realm provides the device identifier when executing any RSI_RDEV command.

See also:

- [A9.2.2 Mapping from virtual device ID to VDEV object](#)
- [B4.3.52 RMI_VDEV_CREATE command](#)

A9.4.2 Virtual device invariants

R₀₂₀₂ Providing a tdi_id which is already used by another VDEV within the same segment causes RMI_VDEV_CREATE to fail.

U₀₂₀₃ The RMM can track usage of tdi_id values within a segment by using SW_RESERVED bits in the SMMU Stream Table Entry.

R₀₂₀₄ Providing a tdi_id which is outside the RID range of the parent PDEV causes RMI_VDEV_CREATE to fail.

A9.4.3 Virtual device lifecycle

A9.4.3.1 States

D₀₂₀₅

The states of a VDEV are listed below.

State	Description
VDEV_NEW	Initial state of the device.
VDEV_READY	No device transaction is associated with the VDEV.
VDEV_COMMUNICATING	The RMM is communicating with the VDEV.
VDEV_STOPPING	The RMM is communicating with the VDEV to stop the device interface.
VDEV_STOPPED	Device interface is stopped.
VDEV_ERROR	Device interface has reported a fatal error.

A9.4.3.2 State transitions

I₀₂₀₆

Permitted VDEV state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of a VDEV. A transition to the pseudo-state *NULL* represents destruction of a VDEV.

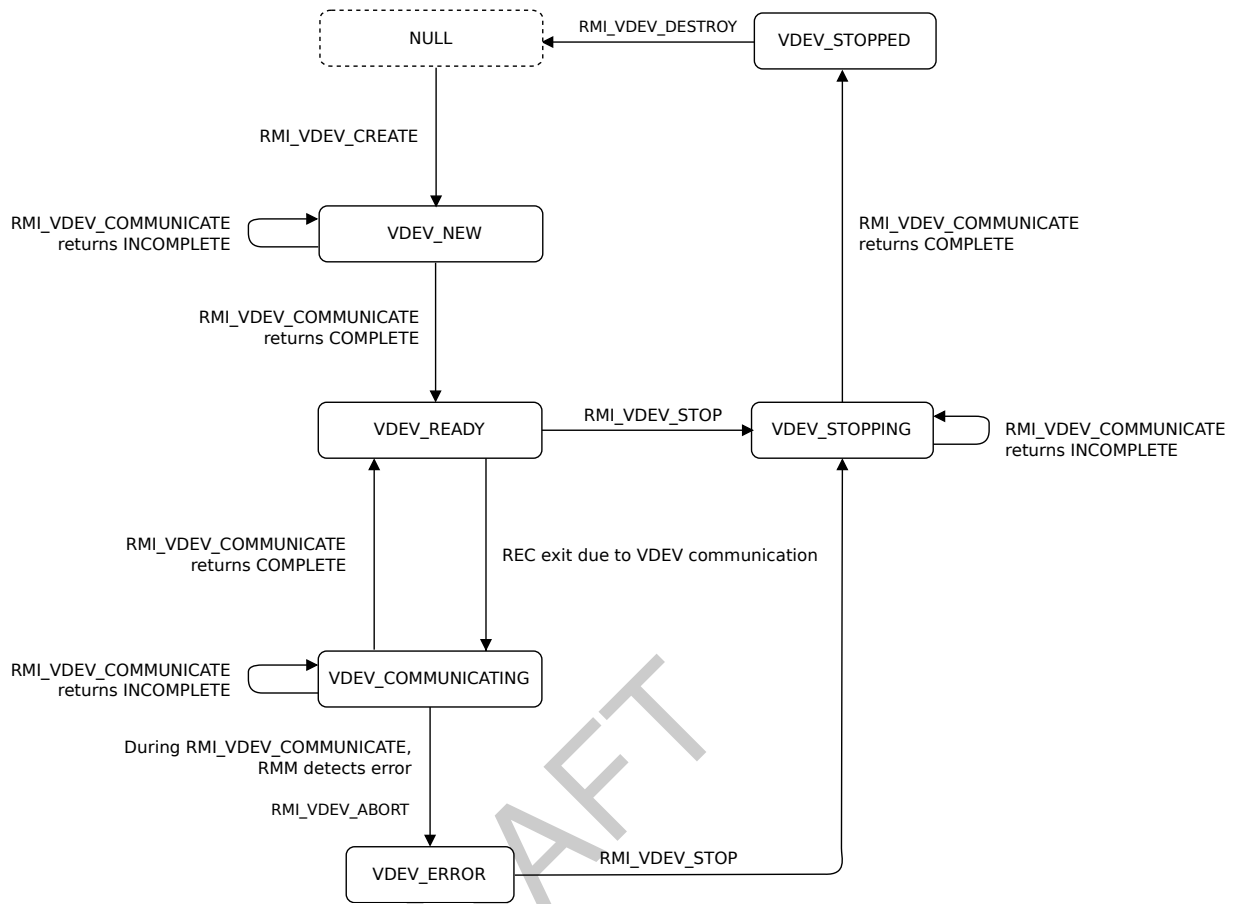


Figure A9.3: VDEV state transitions

- I₀₂₀₇ While a VDEV is in VDEV_NEW state, execution of RMI_PDEV_COMMUNICATE causes the RMM to initialize the device.
- Once device initialization is complete, the VDEV moves to VDEV_READY state.
- I₀₂₀₈ While a VDEV is in VDEV_READY state, on a REC exit due to VDEV communication, the VDEV moves to VDEV_COMMUNICATING state.
- I₀₂₀₉ While a VDEV is in VDEV_COMMUNICATING state, the Host can either:
- Transfer device requests and device responses by executing RMI_VDEV_COMMUNICATE. If the return value indicates that the device transaction is complete then the VDEV moves to VDEV_READY state.
 - Abort the device transaction by executing RMI_VDEV_ABORT. On successful execution of this command, the VDEV moves to VDEV_ERROR state.
- I₀₂₁₀ On execution of RMI_VDEV_COMMUNICATE, if the RMM detects a fatal error (such as an unexpected response or a protocol error) then the VDEV moves to VDEV_ERROR state.
- I₀₂₁₁ While a VDEV is in VDEV_READY state or VDEV_ERROR state, the Host can request the RMM to stop the device by executing RMI_VDEV_STOP.
- If the return value is RMI_SUCCESS then the VDEV moves to VDEV_STOPPING state.
- I₀₂₁₂ While a VDEV is in VDEV_STOPPING state, the Host can enact the “stop” device transaction by executing RMI_VDEV_COMMUNICATE.
- If the return value indicates that the device transaction is complete then the VDEV moves to VDEV_STOPPED state.

I₀₂₁₃ While a VDEV is in VDEV_STOPPING state, if the device fails to respond or reports an error then the Host can call RMI_VDEV_COMMUNICATE, passing RMI_DEV_COMM_ERROR.

On successful execution of this command, the VDEV moves to VDEV_STOPPED state.

I₀₂₁₄ While a VDEV is in VDEV_STOPPED state, the Host can reclaim resources by executing RMI_VDEV_DESTROY.

I₀₂₁₅ Permitted VDEV state transitions are shown in the following table. The rightmost column lists the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of a VDEV object. A transition to the pseudo-state *NULL* represents destruction of a VDEV object.

From state	To state	Events
<i>NULL</i>	VDEV_NEW	RMI_VDEV_CREATE
VDEV_NEW	VDEV_NEW	RMI_VDEV_ABORT RMI_VDEV_COMMUNICATE
VDEV_NEW	VDEV_READY	RMI_VDEV_COMMUNICATE
VDEV_READY	VDEV_COMMUNICATING	REC exit due to VDEV communication
VDEV_READY	VDEV_STOPPING	RMI_VDEV_STOP
VDEV_COMMUNICATING	VDEV_COMMUNICATING	RMI_VDEV_COMMUNICATE
VDEV_COMMUNICATING	VDEV_ERROR	RMI_VDEV_ABORT
VDEV_ERROR	VDEV_STOPPING	RMI_VDEV_STOP
VDEV_STOPPING	VDEV_STOPPING	RMI_VDEV_COMMUNICATE
VDEV_STOPPING	VDEV_STOPPED	RMI_VDEV_COMMUNICATE
VDEV_STOPPED	<i>NULL</i>	RMI_VDEV_DESTROY

See also:

- [B4.3.48 RMI_VDEV_ABORT command](#)
- [B4.3.50 RMI_VDEV_COMMUNICATE command](#)
- [B4.3.52 RMI_VDEV_CREATE command](#)
- [B4.3.53 RMI_VDEV_DESTROY command](#)
- [B4.3.55 RMI_VDEV_STOP command](#)

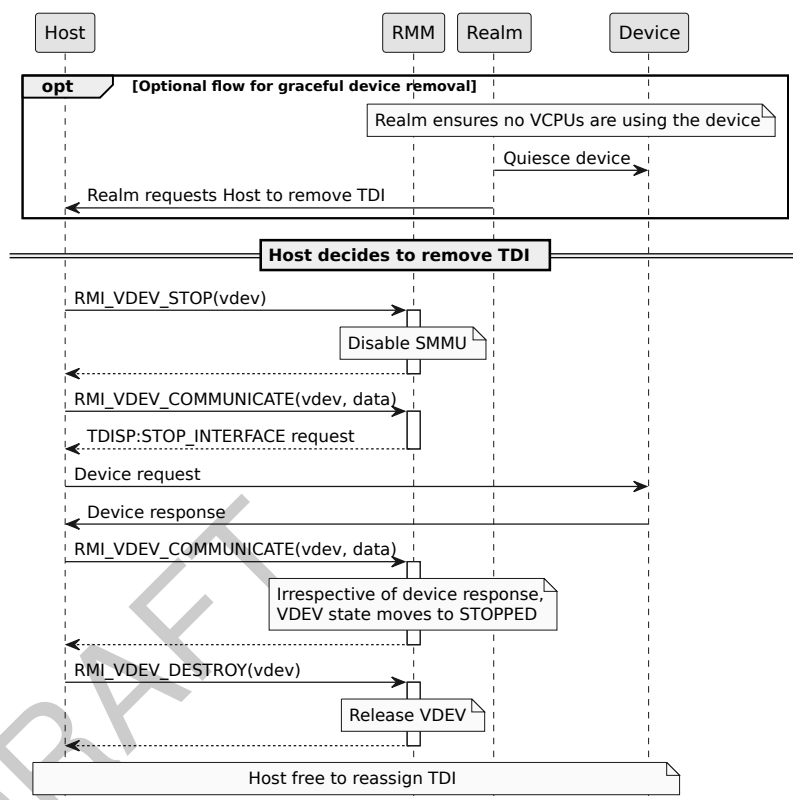
See also:

- [A4.3.13 REC exit due to VDEV communication](#)

A9.4.4 Virtual device flows

A9.4.4.1 Virtual device teardown flow

I₀₂₁₆ Teardown of a VDEV is illustrated in the following sequence diagram.



A9.5 Realm management of an assigned device interface

This section describes interaction between a Realm and the RMM to manage an RDEV.

A9.5.1 Interruptible Realm device operations

D₀₂₁₇ The following are *interruptible Realm device operations*:

- Requesting digests of device attestation evidence
- Requesting a device interface report
- Requesting device measurements
- Locking a device interface
- Starting a device interface
- Stopping a device interface

I₀₂₁₈ Interruptible Realm device operation are:

- Initiated by the Realm executing RSI commands
- Enacted via a REC exit due to VDEV communication, and by subsequent Host calls to RMI_VDEV_COMMUNICATE.

I₀₂₁₉ The Realm programming model for interruptible Realm device operation consists of two phases:

1. The Realm initiates the operation by executing an RSI command. Successful execution of this command causes the RDEV to transition to a *busy* state.
2. The Realm executes RSI_RDEV_CONTINUE in a loop, until the result is not RSI_INCOMPLETE.

The following pseudocode illustrates this programming model, using RSI_RDEV_GET_INTERFACE_REPORT as an example.

```
int get_interface_report(...)
{
    int ret;

    ret = RSI_RDEV_GET_INTERFACE_REPORT(dev_id, ...);
    if (ret) {
        return ret;
    }

    do {
        ret = RSI_RDEV_CONTINUE(dev_id);
    } while (ret == RSI_INCOMPLETE);

    return ret;
}
```

I₀₂₂₀ Once an interruptible Realm device operation has been initiated by the Realm, it must either be:

- Completed by the Realm, via execution of RSI_RDEV_CONTINUE and subsequent calls by the Host to RMI_VDEV_COMMUNICATE, or
- Aborted by the Host, via execution of RMI_VDEV_ABORT.

If the Realm wishes to abort an interruptible Realm device operation, it must request the Host to do so.

I₀₂₂₁ The set of commands which can initiate an interruptible Realm device operation are as follows:

- RSI_RDEV_GET_INTERFACE_REPORT
- RSI_RDEV_GET_MEASUREMENTS
- RSI_RDEV_LOCK
- RSI_RDEV_START
- RSI_RDEV_STOP

See also:

- [A4.3.13 REC exit due to VDEV communication](#)
- [A9.2 Communication between RMM and a device](#)
- [B4.3.50 RMI_VDEV_COMMUNICATE command](#)
- [B5.3.18 RSI_RDEV_GET_INTERFACE_REPORT command](#)
- [B5.3.19 RSI_RDEV_GET_MEASUREMENTS command](#)
- [B5.3.21 RSI_RDEV_LOCK command](#)
- [B5.3.22 RSI_RDEV_START command](#)
- [B5.3.23 RSI_RDEV_STOP command](#)

A9.5.2 Realm retrieval of device attestation evidence

I₀₂₂₂ A Realm is expected to retrieve cached device attestation evidence from the Host via an RSI_HOST_CALL interface. Details of this interface are out of scope of this specification.

I₀₂₂₃ A Realm can retrieve digests of cached device attestation evidence from the RMM by executing the RSI_RDEV_GET_INFO command.

See also:

- [A9.2.4 Host caching of device attestation evidence](#)
- [B5.3.4 RSI_HOST_CALL command](#)
- [B5.3.16 RSI_RDEV_GET_INFO command](#)

A9.5.3 Realm validation of device memory mappings

I₀₂₂₄ A Realm device interface report describes the memory regions of the device. Each device memory region has the following attributes in the report:

- PA base address of the region.
This is obfuscated by addition of a random offset value to the system physical address. The offset value is known to the RMM.
- Size of the region.
- Whether the output address of the mapping is within the system coherent memory space.

I₀₂₂₅ A Realm can validate by execution of RSI_RDEV_VALIDATE_MAPPING that each MMIO region in the Realm device interface report has been correctly mapped into the Realm's Protected IPA space.

I₀₂₂₆ PCIe classifies memory locations into the following categories:

- “TEE memory”: must have mechanisms to ensure confidentiality of data. May additionally provide integrity properties on the data. In a CCA system, this category corresponds to memory locations within Realm PAS.
- “Non-TEE memory”: assumed not to have mechanisms to ensure confidentiality or integrity of data. In a CCA system, this category corresponds to memory locations in Non-Secure PAS.

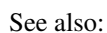
I₀₂₂₇ Arm recommends that RSI_RDEV_VALIDATE_MAPPING is only called for TEE memory device locations. The RMM does not prevent validation of mappings to Non-TEE memory device locations. However, because all accesses to device memory via Protected IPA space are made with the IDE T-bit set to 1, access to a Non-TEE memory location may be rejected by the device. For further details, refer to [PCI Express 6.0 specification \[14\]](#).

I₀₂₂₈ Execution of RSI_RDEV_VALIDATE_MAPPING initiates a Device memory mapping validation request. On execution of RSI_RDEV_VALIDATE_MAPPING, the RMM records the PA base address of the region and the flags provided by the Realm in the REC. On subsequent execution of RMI_RTT_DEV_MEM_VALIDATE, the RMM validates that the contents of the target RTTE are compatible with the PA base address of the region and the flags provided by the Realm. If this validation passes then the RMM sets the RIPAS of the RTTE to DEV.

- R₀₂₂₉ The RTTE attributes checked by RMI_RTT_DEV_MEM_VALIDATE, when the target RIPAS is RIPAS_DEV, are as follows:
- The IPA to PA mapping is consistent with the PA base address of the region provided by the Realm.
 - The memory attributes are consistent with the value of the “coh” flag provided by the Realm.
 - The system memory space (coherent or non-coherent) of the PA is consistent with the “coh” flag provided by the Realm.
 - The limited ordering properties of the PA (LOR or non-LOR) are consistent with the “order” flag provided by the Realm.
- I₀₂₃₀ Creation and validation of device memory mappings is illustrated in the following sequence diagram.

DRAFT

A9.5. Realm management of an assigned device interface



- #### A9.5.4 Realm device attributes

D₀₂₃₁

The attributes of an RDEV are summarized in the following table.

Name	Type	Description
state	RmmRdevState	Lifecycle state
operation	RmmRdevOperation	Operation being performed by the RDEV
vdev_ptr	Address	PA of VDEV associated with this RDEV

A9.5.5 Realm device lifecycle

A9.5.5.1 States

D₀₂₃₂

The states of a RDEV are listed below.

State	Description
RDEV_UNLOCKED	Device interface is unlocked.
RDEV_UNLOCKED_BUSY	Device interface is unlocked and is handling an interruptible Realm device operation.
RDEV_LOCKED	Device interface is locked.
RDEV_LOCKED_BUSY	Device interface is locked and is handling an interruptible Realm device operation.
RDEV_STARTED	Device interface is started.
RDEV_STARTED_BUSY	Device interface is started and is handling an interruptible Realm device operation.
RDEV_STOPPING	Device interface is stopping.
RDEV_STOPPED	Device interface is stopped.
RDEV_ERROR	Device interface has reported a fatal error.

A9.5.5.2 State transitions

I₀₂₃₃

Permitted RDEV Realm state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

A transition from the pseudo-state *NULL* represents creation of an RDEV.

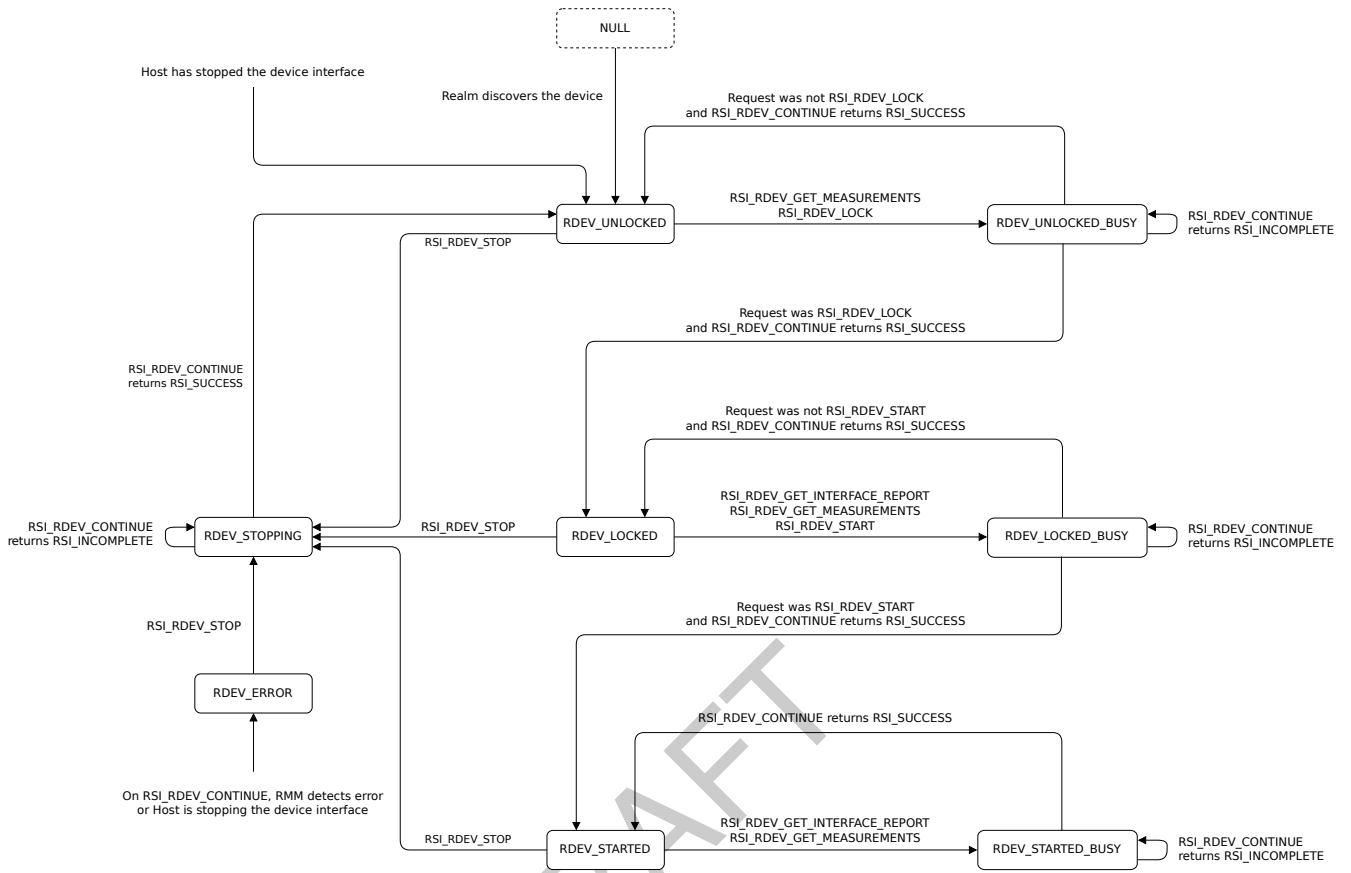


Figure A9.4: RDEV state transitions

I₀₂₃₄ The Realm discovers an assigned PCIe TDI by probing PCIe config space. Arm expects that this will be emulated by the Host, via Unprotected IPA space.

I₀₂₃₅ While an RDEV is in RDEV_UNLOCKED state, the Realm can execute any of the following commands:

- RSI_RDEV_GET_MEASUREMENTS
- RSI_RDEV_LOCK

On successful execution of this command, the RDEV moves to RDEV_UNLOCKED_BUSY state and records the requested operation.

X₀₂₃₆ Permitting RSI_RDEV_GET_MEASUREMENTS while the RDEV is in RDEV_UNLOCKED state allows the Realm to perform the following sequence:

1. Request a measurement of the initial firmware present in a device.
2. Transition the device to RDEV_LOCKED.
3. Upload new or additional firmware to the device.
4. Verify that the firmware has been successfully uploaded, by requesting a new measurement from the device.

I₀₂₃₇ While an RDEV is in RDEV_UNLOCKED_BUSY state, the Realm can execute RSI_RDEV_CONTINUE.

Once the requested operation is complete:

- If the requested operation was RSI_RDEV_LOCK, the RDEV moves to RDEV_LOCKED state.
- Otherwise the RDEV moves to RDEV_UNLOCKED state.

I₀₂₃₈ While an RDEV is in RDEV_LOCKED state, the Realm can execute any of the following commands:

- RSI_RDEV_GET_INTERFACE_REPORT
- RSI_RDEV_GET_MEASUREMENTS
- RSI_RDEV_START

On successful execution of this command, the RDEV moves to RDEV_LOCKED_BUSY state and records the requested operation.

I₀₂₃₉ While an RDEV is in RDEV_LOCKED_BUSY state, the Realm can execute RSI_RDEV_CONTINUE.

Once the requested operation is complete:

- If the requested operation was RSI_RDEV_START, the RDEV moves to RDEV_STARTED state.
- Otherwise the RDEV moves to RDEV_LOCKED state.

I₀₂₄₀ While an RDEV is in RDEV_STARTED state, the Realm can execute any of the following commands:

- RSI_RDEV_GET_INTERFACE_REPORT
- RSI_RDEV_GET_MEASUREMENTS

On successful execution of this command, the RDEV moves to RDEV_STARTED_BUSY state and records the requested operation.

I₀₂₄₁ While an RDEV is in RDEV_STARTED_BUSY state, the Realm can execute RSI_RDEV_CONTINUE.

Once the requested operation is complete, the RDEV moves to RDEV_STARTED state.

I₀₂₄₂ On execution of RSI_RDEV_CONTINUE, if the RMM detects a fatal error (such as an unexpected response or a protocol error) then the RDEV moves to RDEV_ERROR state.

I₀₂₄₃ While an RDEV is in any state, the Realm can execute RSI_RDEV_STOP.

On successful execution of this command the RDEV moves to RDEV_STOPPING state.

Following successful execution of this command, accesses from the device to Realm memory are blocked.

I₀₂₄₄ While an RDEV is in RDEV_STOPPING state, the Realm can execute RSI_RDEV_CONTINUE.

Once the requested operation is complete, the RDEV moves to RDEV_STOPPED state.

I₀₂₄₅ Permitted RDEV state transitions are shown in the following table. The rightmost column lists the events which can cause the corresponding state transition.

From state	To state	Events
RDEV_UNLOCKED	RDEV_UNLOCKED_BUSY	RSI_RDEV_GET_MEASUREMENTS RSI_RDEV_LOCK
RDEV_UNLOCKED_BUSY	RDEV_UNLOCKED_BUSY	RSI_RDEV_CONTINUE
RDEV_UNLOCKED_BUSY	RDEV_LOCKED	RSI_RDEV_CONTINUE
RDEV_LOCKED	RDEV_LOCKED_BUSY	RSI_RDEV_GET_INTERFACE_REPORT RSI_RDEV_GET_MEASUREMENTS RSI_RDEV_START
RDEV_LOCKED_BUSY	RDEV_LOCKED_BUSY	RSI_RDEV_CONTINUE
RDEV_LOCKED_BUSY	RDEV_STARTED	RSI_RDEV_CONTINUE
RDEV_STARTED	RDEV_STARTED_BUSY	RSI_RDEV_GET_INTERFACE_REPORT RSI_RDEV_GET_MEASUREMENTS
RDEV_STARTED_BUSY	RDEV_STARTED_BUSY	RSI_RDEV_CONTINUE
RDEV_UNLOCKED	RDEV_STOPPING	RSI_RDEV_STOP
RDEV_LOCKED	RDEV_STOPPING	RSI_RDEV_STOP

From state	To state	Events
RDEV_STARTED	RDEV_STOPPING	RSI_RDEV_STOP
RDEV_UNLOCKED	RDEV_ERROR	RSI_RDEV_CONTINUE Host is stopping device interface
RDEV_UNLOCKED_BUSY	RDEV_ERROR	RSI_RDEV_CONTINUE Host is stopping device interface
RDEV_UNLOCKED_BUSY	RDEV_UNLOCKED	RSI_RDEV_CONTINUE Host has stopped device interface
RDEV_LOCKED	RDEV_ERROR	RSI_RDEV_CONTINUE Host is stopping device interface
RDEV_LOCKED_BUSY	RDEV_ERROR	RSI_RDEV_CONTINUE Host is stopping device interface
RDEV_LOCKED_BUSY	RDEV_UNLOCKED	RSI_RDEV_CONTINUE Host has stopped device interface
RDEV_STARTED	RDEV_ERROR	RSI_RDEV_CONTINUE Host is stopping device interface
RDEV_STARTED_BUSY	RDEV_ERROR	RSI_RDEV_CONTINUE Host is stopping device interface
RDEV_STARTED_BUSY	RDEV_UNLOCKED	RSI_RDEV_CONTINUE Host has stopped device interface
RDEV_ERROR	RDEV_STOPPING	RSI_RDEV_STOP
RDEV_STOPPING	RDEV_STOPPING	RSI_RDEV_CONTINUE
RDEV_STOPPING	RDEV_UNLOCKED	RSI_RDEV_CONTINUE

See also:

- [B5.3.15 RSI_RDEV_CONTINUE command](#)
- [B5.3.18 RSI_RDEV_GET_INTERFACE_REPORT command](#)
- [B5.3.19 RSI_RDEV_GET_MEASUREMENTS command](#)
- [B5.3.21 RSI_RDEV_LOCK command](#)
- [B5.3.22 RSI_RDEV_START command](#)
- [B5.3.23 RSI_RDEV_STOP command](#)

A9.5.5.3 Relationship between RDEV state and TDISP TDI state

I₀₂₄₆ The lifecycle of an RDEV closely resembles the lifecycle of a TDISP TDI. There cannot exist a direct mapping between the two, because changes in TDI state (for example due to Host action) may not be immediately observed by either the RMM or the Realm. However, the two states are sufficiently closely coupled to provide the Realm with important guarantees regarding the TDI state.

R₀₂₄₇ On transition of an RDEV to RDEV_LOCKED state, the corresponding TDI transitions to CONFIG_LOCKED state.

R₀₂₄₈ On transition of an RDEV to RDEV_STARTED state, the corresponding TDI transitions to RUN state.

R₀₂₄₉ On detection by the RMM that a TDI is in ERROR state, the corresponding RDEV transitions to RDEV_ERROR state.

See also:

- [PCI Express 6.0 specification \[14\]](#)

A9.5.5.4 Relationship between RDEV state and SMMU enablement

R0250

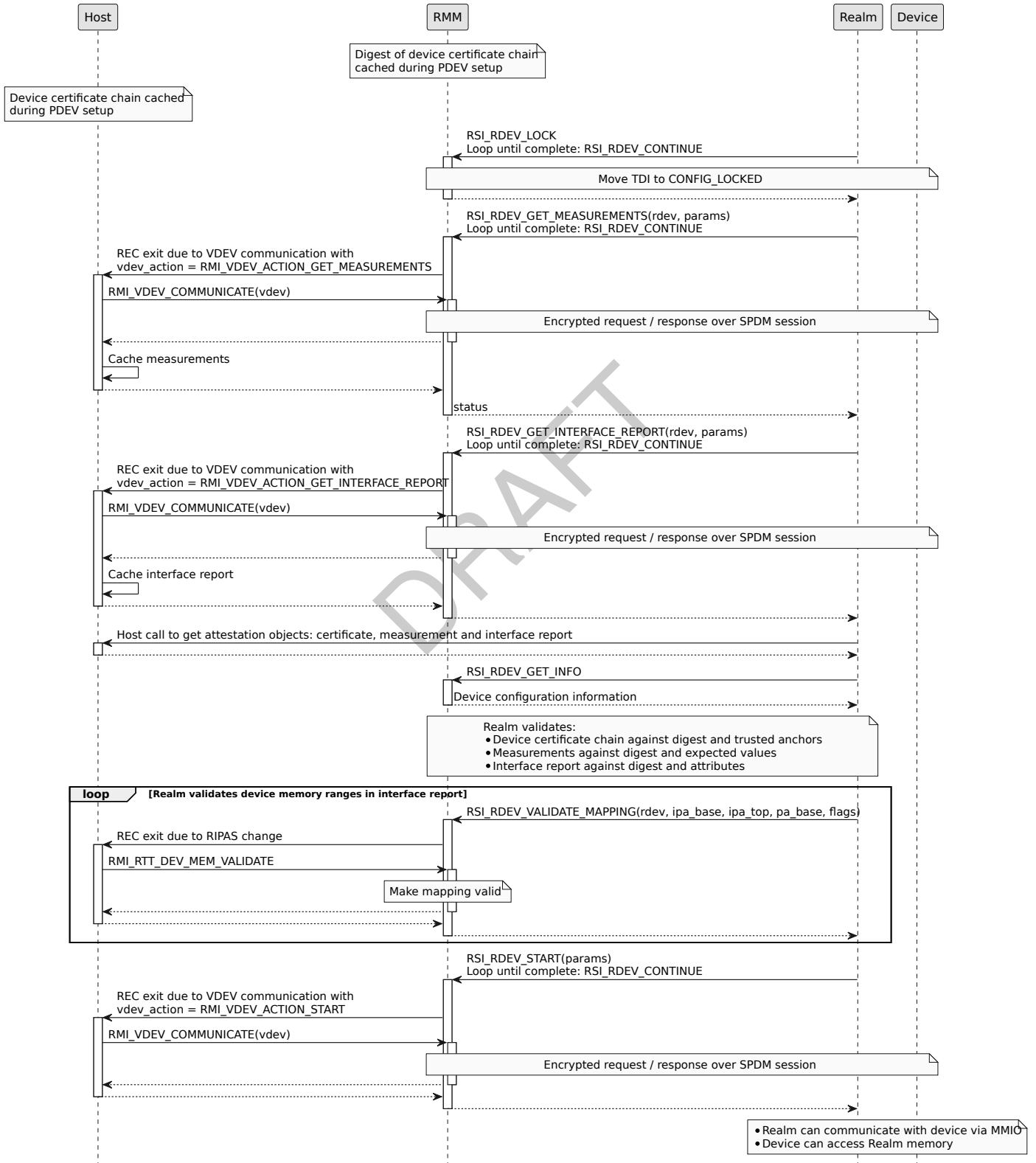
When the state of an RDEV is RDEV_STARTED, the SMMU permits traffic from the device to the Realm's Protected IPA space. When the state of an RDEV is not RDEV_STARTED, the SMMU blocks traffic from the device to the Realm's Protected IPA space.

DRAFT

A9.5.6 Realm device flows

A9.5.6.1 Realm device setup flow

Setup of an RDEV is illustrated in the following sequence diagram.



A9.6 Virtual SMMU

- D₀₂₅₂ A *Virtual SMMU* (VSMMU) object stores the state of a Arm VSMMU which is emulated by the RMM.
- I₀₂₅₃ A physical device which contains functions that can be assigned to Realms must access memory via a physical SMMU (PSMMU).
- I₀₂₅₄ A PSMMU is identified by its base physical address.
- I₀₂₅₅ For every PSMMU used by one or more device functions which are assigned to a Realm, if one or more of those device functions require stage 1 translation then a corresponding VSMMU must be created.
- I₀₂₅₆ A VSMMU can only be created while the Realm state is NEW.

See also:

- [Arm System Memory Management Unit Architecture Specification \[16\]](#)

A9.6.1 VSMMU lifecycle

- I₀₂₅₇ A VSMMU is created and bound to a Realm by execution of RMI_VSMMU_CREATE. This command only permits VSMMU creation while the Realm state is NEW.
- I₀₂₅₈ A VSMMU is destroyed by execution of RMI_VSMMU_DESTROY.
- I₀₂₅₉ A VSMMU is bound to a VDEV by execution of RMI_VDEV_CREATE, with RmiVdevFlags::VSMMU set.
- I₀₂₆₀ A VSMMU is unbound from a VDEV by execution of RMI_VDEV_DESTROY.
- I₀₂₆₁ Mappings from Realm Protected IPA space to a VSMMU object are created by execution of RMI_VSMMU_MAP. This command only permits such mappings to be created within the register IPA range specified on creation of the VSMMU object. This causes the HIPAS to change from UNASSIGNED to ASSIGNED_VSMMU.
- I₀₂₆₂ Mappings from Realm Protected IPA space to a VSMMU object are removed by execution of RMI_VSMMU_UNMAP.

See also:

- [A9.6.3 VSMMU liveness](#)
- [A9.6.4 VSMMU validation](#)
- [B4.3.52 RMI_VDEV_CREATE command](#)
- [B4.3.53 RMI_VDEV_DESTROY command](#)
- [B4.3.57 RMI_VSMMU_CREATE command](#)
- [B4.3.58 RMI_VSMMU_DESTROY command](#)
- [B4.3.59 RMI_VSMMU_MAP command](#)
- [B4.3.60 RMI_VSMMU_UNMAP command](#)

A9.6.2 VSMMU attributes

- D₀₂₆₃ The attributes of a VSMMU are summarized in the following table.

Name	Type	Description
realm	Address	PA of RD of Realm which owns this VSMMU
reg_base	Address	Base IPA of register base in Realm's Protected IPA space
reg_top	Address	Top IPA of register base in Realm's Protected IPA space
aidr	Bits64	SMMU_AIDR register value
idr	Bits64 [7]	SMMU_IDR register values

I₀₂₆₄ The Host provides the VSMMU register IPA range when executing RMI_VSMMU_CREATE.

I₀₂₆₅ The Host provides the VSMMU AIDR and IDR values when executing RMI_VSMMU_CREATE.

See also:

- [B4.3.57 RMI_VSMMU_CREATE command](#)

A9.6.3 VSMMU liveness

D₀₂₆₆ VSMMU liveness is a property which means that there exists one or more mappings from Realm Protected IPA space to the VSMMU object.

I₀₂₆₇ If a VSMMU is live, it cannot be destroyed.

See also:

- [A9.6.1 VSMMU lifecycle](#)
- [B3.161 VsmmuIsLive function](#)
- [B4.3.58 RMI_VSMMU_DESTROY command](#)

A9.6.4 VSMMU validation

I₀₂₆₈ The Realm queries whether an IPA is the base address of a VSMMU by execution of RSI_VSMMU_GET_INFO.

I₀₂₆₉ The Realm activates the register interface of a VSMMU by execution of RSI_VSMMU_ACTIVATE. This causes the RIPAS of the IPA range to change from EMPTY to DEV.

I₀₂₇₀ The attributes of a VSMMU are guaranteed not to change between execution of RSI_VSMMU_GET_INFO and RSI_VSMMU_ACTIVATE.

I₀₂₇₁ The programming model for validating and activating a VSMMU is shown in the following pseudocode:

```
int realm_validate_vsmmu(unsigned long base, unsigned long top)
{
    unsigned long new_base, response;
    int ret = RSI_SUCCESS;

    // Check whether "base" identifies a VSMMU
    ret = rsi_vsmmu_get_info(base);
    if (ret != RSI_SUCCESS) {
        return ret;
    }

    // Activate the VSMMU register interface
    while (base != top) {
        ret = rsi_vsmmu_activate(base, top, &new_base);

        if (ret != RSI_SUCCESS) {
            return ret;
        }

        base = new_base;
    }

    return RSI_SUCCESS;
}
```

See also:

- [A9.6.1 VSMMU lifecycle](#)
- [B5.3.27 RSI_VSMMU_ACTIVATE command](#)

- [B5.3.28 RSI_VSMMU_GET_INFO command](#)

A9.6.5 PSMMU interrupts

- I₀₂₇₂ For a PSMMU with MSI, configuration is provided by execution of RMI_PSMMU_MSI_CONFIG.
- I₀₂₇₃ For a PSMMU if wired interrupts are supported, no configuration is required from the RMM side. It is assumed that the RMM implementation will initialise SMMU_*IRQ_CFG0 registers to 0 to allow wired interrupts
- I₀₂₇₄ On taking a physical SMMU interrupt, the Host notifies the RMM by execution of RMI_PSMMU_IRQ_NOTIFY. In response, the RMM may request the Host to inject a given virtual MSI into a specified Realm.

See also:

- [A9.6.1 VSMMU lifecycle](#)
- [B4.3.22 RMI_PSMMU_IRQ_NOTIFY command](#)
- [B4.3.23 RMI_PSMMU_MSI_CONFIG command](#)

DRAFT

A9.7 Device access to a Protected IPA

I₀₂₇₅ Device access to a Protected IPA follows the same rules as Realm data access to a Protected IPA.

See also:

- [A5.2.3 Realm access to a Protected IPA](#)

DRAFT

Chapter A10

Planes

This section describes how a Realm can be divided into multiple mutually isolated execution environments, called Planes.

Provisional

Decide whether this section should be:

- Moved to the [Concepts](#) chapter, alongside the [Realm](#) section.
- Left in this position, on the grounds that, like [Realm device assignment](#), Planes is an optional feature.

A10.1 Planes overview

Provisional

The following aspects and implications of Planes will be described in a future release of this specification:

- Access from P_n to a device assigned to the Realm

D₀₂₇₆

A Realm contains:

- a single *primary Plane*
- zero or more *auxiliary Planes*.

A Realm with a non-zero number of auxiliary Planes is said to contain *multiple Planes*.

I₀₂₇₇

The number of auxiliary Planes is specified by the Host at Realm creation.

D₀₂₇₈

Planes within a Realm are identified using a zero-based *Plane index*.

The Plane index of the primary Plane is zero.

D₀₂₇₉

When referring to the primary Plane of a Realm, this specification uses the term *Plane 0*, or *P0*.

When referring to any auxiliary Plane, this specification uses the term *P_n*.

I₀₂₈₀

All Planes within a Realm share a single IPA space.

Stage 2 memory access permissions for a given IPA can differ between Planes.

I₀₂₈₁

Each Plane has a VMID which is unique both within the owning Realm and among all Realms.

I₀₂₈₂

A Realm with multiple Planes may either have:

- An RTT tree per Plane, or
- A single RTT tree, with per-Plane access permissions being managed indirectly.

I₀₂₈₃

On REC exit due to Data Abort or Instruction Abort, the index of the RTT tree used by the exited Plane is provided to the Host.

This allows the Host to know which RTT tree must be modified in order to service a fault.

D₀₂₈₄

A given VPE executes in one Plane at a time.

This is referred to as the *active Plane* of the VPE.

I₀₂₈₅

The following capabilities are available only to P0:

- Change the active Plane of the current VPE
- Read and write register state of other Planes in the current VPE
- Configure and take traps from other Planes in the current VPE
- Control delivery of virtual interrupts to other Planes in the current VPE

See also:

- [A3.11 Support for auxiliary Planes](#)
- [A10.2 Planes exception model](#)
- [A10.3 Planes memory management](#)
- [A10.4 Planes interrupts](#)

A10.2 Planes exception model

Provisional

Decide whether this section should be integrated into the main [Realm exception model](#) chapter.

A10.2.1 Plane exception model overview

- D₀₂₈₆ A *Plane entry* is a transition from P0 to Pn, due to execution of RSI_PLANE_ENTER.
- I₀₂₈₇ P0 provides the index of the target Plane (Pn) as an input to the RSI_PLANE_ENTER command.
- D₀₂₈₈ A *Plane exit* is return to P0 from an execution of RSI_PLANE_ENTER which caused a Plane entry.
- D₀₂₈₉ A *PlaneRun* object is a data structure used to pass values between the RMM and P0 on Plane entry and on Plane exit.
- I₀₂₉₀ A PlaneRun object is stored in Realm memory.
- I₀₂₉₁ Between a Plane entry and a Plane exit, a REC exit and REC entry may occur.

As an example:

1. Running in REC A, P0 executes RSI_PLANE_ENTER, passing target Plane index 1.
This causes a Realm exit to the RMM, followed by a Realm entry to P1, within REC A.
2. Running in REC A, P1 accesses an IPA which is not mapped (HIPAS = UNASSIGNED, RIPAS = RAM).
This causes a REC exit to the Host.
3. The Host executes RMI_REC_ENTER, passing the address of REC A.
This causes the RMM to return to P1 within REC A.
4. Running in REC A, P1 accesses an IPA whose RIPAS is EMPTY.
This causes a Plane exit to P0.

I₀₂₉₂ Following a REC exit from P0, on the next entry to the same REC, control returns to P0.

I₀₂₉₃ Following a REC exit from Pn, on the next entry to the same REC, the Host determines whether:

- Control returns to Pn. This is the default behaviour.
- A Plane exit occurs, returning control to P0. This can be triggered for example due to the Host injecting a virtual interrupt into the REC.

See also:

- [A4.1 Realm exception model overview](#)
- [B5.3.12 RSI_PLANE_ENTER command](#)
- [B5.4.19 RsiPlaneRun type](#)

A10.2.2 Plane entry

- D₀₂₉₄ An *RsiPlaneEnter object* is a data structure used to pass values from P0 to the RMM on Plane entry.
- I₀₂₉₅ An RsiPlaneEnter object is stored in the RsiPlaneRun object which is passed by P0 as an input to the RSI_PLANE_ENTER command.
- I₀₂₉₆ In this chapter, both `plane_enter` and “the RsiPlaneEnter object” refer to the RsiPlaneEnter object which is provided to the RSI_PLANE_ENTER command.
- I₀₂₉₇ On Plane entry, execution state is restored from the RsiPlaneEnter object to the PE.

I₀₂₉₈ An RsiPlaneEnter object contains attributes which are used to manage Pn virtual interrupts.

D₀₂₉₉ The attributes of an RsiPlaneEnter object are summarized in the following table.

Name	Byte offset	Type	Description
flags	0x0	RsiPlaneEnterFlags	Flags
pc	0x8	Bits64	Program counter
gprs[31]	0x100	Bits64	Registers
gicv3_hcr	0x200	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0x208	Bits64	GICv3 List Register values

A10.2.3 Plane exit

D₀₃₀₀ An RsiPlaneExit object is a data structure used to pass values from the RMM to P0 on Plane exit.

I₀₃₀₁ An RsiPlaneExit object is stored in the RsiPlaneRun object which is passed by P0 as an input to the RSI_PLANE_ENTER command.

I₀₃₀₂ In this chapter, both plane_exit and “the RsiPlaneExit object” refer to the RsiPlaneExit object which is provided to the RSI_PLANE_ENTER command.

I₀₃₀₃ On Plane exit, execution state is saved from the PE to the RsiPlaneExit object.

D₀₃₀₄ The attributes of an RsiPlaneExit object are summarized in the following table.

Name	Byte offset	Type	Description
reason	0x0	RsiPlaneExitReason	Exit reason
elr_el2	0x100	Bits64	Exception Link Register
esr_el2	0x108	Bits64	Exception Syndrome Register
far_el2	0x110	Bits64	Fault Address Register
hpfar_el2	0x118	Bits64	Hypervisor IPA Fault Address register
gprs[31]	0x200	Bits64	Registers
gicv3_hcr	0x300	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0x308	Bits64	GICv3 List Register values
gicv3_misr	0x388	Bits64	GICv3 Maintenance Interrupt State Register value
gicv3_vmcr	0x390	Bits64	GICv3 Virtual Machine Control Register value
cntp_ctl	0x400	Bits64	Counter-timer Physical Timer Control Register value
cntp_cval	0x408	Bits64	Counter-timer Physical Timer CompareValue Register value
cntv_ctl	0x410	Bits64	Counter-timer Virtual Timer Control Register value

Name	Byte offset	Type	Description
cntv_cval	0x418	Bits64	Counter-timer Virtual Timer CompareValue Register value

I₀₃₀₅ RsiPlaneExit uses architectural encodings (of ESR, FAR, HPFAR) which are normally observable only to EL2; however, the exit is taken to P0 at EL1. This is justified on the grounds that P0 exists essentially in order to allow part of the job of the hypervisor to be performed inside the Realm.

I₀₃₀₆ On Plane exit, all RsiPlaneExit fields are zero unless specified otherwise.

A10.2.3.1 Plane exit due to synchronous exception

R₀₃₀₇ An exception due to any of the following in Pn causes a Plane exit due to Synchronous Exception:

- Trapped WF* instruction execution
- Data Abort at a Protected IPA
 - Permission fault
 - Access to an IPA whose RIPAS is EMPTY
- Instruction Abort at a Protected IPA
 - Permission fault
 - Access to an IPA whose RIPAS is EMPTY
- HVC instruction execution
- RSI_HOST_CALL execution, if `plane_enter.flags.trap_hc == RSI_TRAP`
- Any other SMC instruction execution

R₀₃₀₈ On Plane exit due to Synchronous Exception, all of the following are true:

- `plane_exit.exit_reason` is `RSI_EXIT_SYNC`.
- `plane_exit.esr_el2` contains the value of `ESR_EL2` at the time of the Realm exit.
- If `plane_exit.esr_el2.EC` indicates Data Abort from a lower Exception level and `plane_exit.esr_el2.ISV == 1` then `plane_exit.far_el2` contains the value of `FAR_EL2` at the time of the Realm exit.
- If `plane_exit.esr_el2.EC` indicates Data Abort from a lower Exception level or Instruction Abort from a lower Exception level, `plane_exit.hpfar_el2` contains the value of `HPFAR_EL2` at the time of the Realm exit.

See also:

- [A4.3.9 REC exit due to Host call](#)

A10.2.4 REC exit from Pn

R₀₃₀₉ An exception due to any of the following in Pn cause a REC exit to the Host:

- The following Synchronous Exceptions:
 - Access to an IPA whose RIPAS is DESTROYED
 - Access to an IPA whose HIPAS is UNASSIGNED and whose RIPAS is not EMPTY
 - Synchronous External Abort
- The following Asynchronous Exceptions:
 - IRQ
 - FIQ
 - SError

- RSI_HOST_CALL execution, if `plane_enter.flags.trap_hc == RSI_NO_TRAP`. In this case, the result is a REC exit due to Host call.

I₀₃₁₀ Any other exception during execution of Pn causes a Plane exit to P0.

A10.2.5 Pn execution of HVC and SMC

I₀₃₁₁ On Plane exit due to execution by Pn of an HVC instruction, possible actions taken by P0 include the following:

- Emulate the instruction
- Forward the request to the Host using RSI_HOST_CALL
- Return SMCCC_NOT_SUPPORTED to Pn.

I₀₃₁₂ On Plane exit due to execution by Pn of an RSI command, possible actions taken by P0 include the following:

- Emulate the RSI command
- Return SMCCC_NOT_SUPPORTED to Pn.

See also:

- [Chapter B5 Realm Services Interface](#)
- [B5.3.4 RSI_HOST_CALL command](#)

A10.2.6 Pn system registers

R₀₃₁₃ On Realm creation, all Pn EL0 and EL1 system register values take architecturally-defined reset values.

I₀₃₁₄ P0 can access Pn EL0 and EL1 system register values using the RSI_PLANE_REG_READ and RSI_PLANE_REG_WRITE commands.

See also:

- [B5.3.13 RSI_PLANE_REG_READ command](#)
- [B5.3.14 RSI_PLANE_REG_WRITE command](#)

A10.3 Planes memory management

Provisional

Decide whether this section should be integrated into the main [Realm memory management](#) chapter.

- I₀₃₁₅ All Planes within a Realm have the same IPA size.
- I₀₃₁₆ If a given IPA x is mapped to a given PA y in one Plane, x is not mapped to a different PA z in any other Plane within the Realm.

A10.3.1 Auxiliary RTT

- I₀₃₁₇ A Realm which is configured to have an RTT tree per Plane has one *primary RTT tree* and (number of auxiliary Planes) *auxiliary RTT trees*.
- D₀₃₁₈ For a Realm which is configured to have an RTT tree per Plane, RTT trees are identified using a zero-based *RTT tree index*.
- The RTT tree index of the primary RTT tree is zero.
- D₀₃₁₉ For a Realm which is configured to have an RTT tree per Plane, the mapping from Plane index to RTT tree index is as follows:

Plane index	RTT tree index
0	1
1	2
...	...
n-2	n-1
n-1	0

For a Realm with no auxiliary Planes, $n == 1$ and therefore the index of the single Plane (0) maps to the index of the single RTT tree (0).

- I₀₃₂₀ Creation, destruction and folding of primary RTTs is independent of creation, destruction and folding of auxiliary RTTs for the same IPA range.
- I₀₃₂₁ If a primary RTT entry is live and its state is not TABLE then the Host can create a mapping to the same output address in an auxiliary RTT by executing RMI_RTT_AUX_MAP_PROTECTED or RMI_RTT_AUX_MAP_UNPROTECTED.
- D₀₃₂₂ An IPA is *auxiliary-live* if any of the entries identified by that IPA in auxiliary RTTs are live.
- I₀₃₂₃ In a Realm which is configured to have an RTT tree per Plane, a given Unprotected IPA may be mapped to different output addresses in different Planes.
- U₀₃₂₄ The absence of a rule which states that a given Unprotected IPA must map to the same output address in different Planes avoids the need for the RMM to manage reference counts for NS Granules.
- I₀₃₂₅ If a Protected IPA is auxiliary-live then the corresponding entry in the primary RTT is live.
- This invariant is preserved by blocking any actions which would make the primary RTT entry non-live, including the following:
- RMI_DATA_DESTROY
 - RMI_DEV_MEM_UNMAP

- [RMI_RTT_SET_RIPAS](#)

I₀₃₂₆ If an IPA is auxiliary-live then its RIPAS cannot be changed.

X₀₃₂₇ Specifying that RIPAS is invariant while an IPA is auxiliary-live avoids an implementation of RMI_RTT_SET_RIPAS or RMI_RTT_DEV_MEM_VALIDATE having to walk multiple auxiliary RTT trees.

See also:

- [A5.6.8 RTTE liveness and RTT liveness](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.31 RMI_RTT_AUX_CREATE command](#)
- [B4.3.32 RMI_RTT_AUX_DESTROY command](#)
- [B4.3.33 RMI_RTT_AUX_FOLD command](#)
- [B4.3.34 RMI_RTT_AUX_MAP_PROTECTED command](#)
- [B4.3.35 RMI_RTT_AUX_MAP_UNPROTECTED command](#)
- [B4.3.36 RMI_RTT_AUX_UNMAP_PROTECTED command](#)
- [B4.3.37 RMI_RTT_AUX_UNMAP_UNPROTECTED command](#)
- [B4.3.45 RMI_RTT_SET_RIPAS command](#)

A10.3.2 Stage 2 Access Permissions within a multi-Plane Realm

This section describes how S2AP is controlled within a multi-Plane Realm.

A10.3.3 Stage 2 Access Permissions within a multi-Plane Realm overview

I₀₃₂₈ The S2AP which applies to an access from P_n to a Protected IPA is controlled by P₀.

I₀₃₂₉ The S2AP which applies to an access from P_n to a Protected IPA can be different for each P_n within the Realm.

I₀₃₃₀ An access by P_n to a Protected IPA which violates S2AP causes a Plane exit taken to P₀.

I₀₃₃₁ The S2AP which applies to a Realm access to an Unprotected IPA is the same for all Planes within the Realm.

I₀₃₃₂ A data access by P_n to an Unprotected IPA which violates S2AP causes a REC exit taken to the Host.

I₀₃₃₃ An instruction fetch by P_n to an Unprotected IPA causes a Plane exit taken to P₀.

U₀₃₃₄ On a platform which implements FEAT_S2PIE and FEAT_S2POE, the RMM can utilise these architecture features to store per-Plane S2AP in a single RTT tree.

On a platform which does not implement these architecture features, a separate RTT tree is required for each Plane.

See also:

- [A5.6.11 Stage 2 Access Permissions](#)

A10.3.3.1 Stage 2 Access Permissions for a Protected IPA within a multi-Plane Realm

R₀₃₃₅ The S2AP overlay index of a Protected IPA is between 0 and 14.

I₀₃₃₆ S2AP overlay index 15 is RESERVED.

R₀₃₃₇ At Realm activation, S2AP overlay indices 0 to 14 map to S2AP overlay value *RW+puX* for P₀.

R₀₃₃₈ At Realm activation, S2AP overlay indices 0 to 14 map to S2AP overlay value *NoAccess* for all Planes other than P₀.

D₀₃₃₉ For each S2AP overlay index there is an associated lock bit which applies to S2AP overlay values for Planes other than P₀.

- If the lock bit is LOCKED then the S2AP overlay values for all Planes are immutable.
- If the lock bit is UNLOCKED then the S2AP overlay values for Planes other than P₀ can be changed by P₀.

- R₀₃₄₀ At Realm activation, S2AP overlay index 0 is LOCKED.
- R₀₃₄₁ At Realm activation, S2AP overlay indices 1 to 14 are UNLOCKED.
- R₀₃₄₂ At Realm activation, the S2AP overlay index of all Protected IPAs is 0.
- I₀₃₄₃ The following table summarises the attributes of all S2AP overlay indices for Protected IPA space, at Realm activation.

S2AP overlay index	P0 S2AP overlay value	Pn S2AP overlay values	Lock status for Pn S2AP overlay values
0	<i>RW+puX</i>	<i>NoAccess</i>	LOCKED
1 to 14	<i>RW+puX</i>	<i>NoAccess</i>	UNLOCKED

- I₀₃₄₄ The RSI_MEM_SET_PERM_INDEX command can be used by P0 to change the S2AP overlay index for a Protected IPA.
- I₀₃₄₅ The RSI_MEM_SET_PERM_VALUE command can be used by P0 to change the mapping from a { Plane, S2AP overlay index } tuple to an S2AP overlay value.

See also:

- [Chapter A5 Realm memory management](#)
- [A10.3.1 Auxiliary RTT](#)
- [B5.3.10 RSI_MEM_SET_PERM_INDEX command](#)
- [B5.3.11 RSI_MEM_SET_PERM_VALUE command](#)

A10.3.3.2 Stage 2 Access Permissions change within a multi-Plane Realm

- D₀₃₄₆ An *S2AP change* is a process via which the S2AP of a region of Protected IPA space is changed.
- I₀₃₄₇ An S2AP change consists of actions taken both by P0 within the Realm and by the Host:
- P0 issues an *S2AP change request* by executing RSI_MEM_SET_PERM_INDEX.
 - The input values to this command include:
 - * The requested IPA range: [base, top)
 - * The requested S2AP overlay index
 - * A “cookie”, whose initial value is zero. Across successive calls to RSI_MEM_SET_PERM_INDEX, the cookie is used by the RMM to store an IMPLEMENTATION DEFINED value which tracks progress of the request.
 - The RMM records these values in the REC, and then performs a REC exit due to S2AP change pending.
 - In response, the Host executes zero or more RMI_RTT_SET_S2AP commands.
 - If the requested RIPAS value was not EMPTY then at the next RMI_REC_ENTER the Host can optionally indicate that it rejects the S2AP change request.
- U₀₃₄₈ The purpose of the cookie is to record the progress of the S2AP change request across multiple RTT trees. For a Realm which is configured to use a shared RTT tree, the RMM should return a cookie value of zero.
- X₀₃₄₉ Rejection by the Host of an S2AP change request is intended to be used if the target IPA range extends beyond the agreed DRAM range for the Realm. In this situation, accepting the request may impose unplanned resource costs on the Host, by requiring allocation of additional RTTs.
- I₀₃₅₀ The S2AP change process ensures that a Realm can always reliably determine the maximum S2AP which can be observed at the next access to any Protected IPA.
- I₀₃₅₁ An S2AP change is applied by one or more calls to the RMI_RTT_SET_S2AP command.
- I₀₃₅₂ The order in which the S2AP of Planes and pages within the target IPA range are changed during execution of RSI_MEM_SET_PERM_INDEX is IMPLEMENTATION DEFINED.

I₀₃₅₃ If the input arguments of RSI_MEM_SET_PERM_INDEX are modified by the caller during the loop, it is IMPLEMENTATION DEFINED whether the S2AP of Planes and pages within the target IPA range are changed.

I₀₃₅₄ The P0 programming model for changing S2AP is to call RSI_MEM_SET_PERM_INDEX in a loop until progress reaches the top of the target IPA range, as shown in the following pseudocode:

```
int realm_set_s2ap(unsigned long base, unsigned long top,
                  unsigned int index)
{
    unsigned long new_base, response;
    unsigned long cookie = 0, new_cookie;
    int ret = RSI_SUCCESS;

    while (base != top) {
        ret = rsi_mem_set_perm_index(base, top, index, cookie,
                                    &new_base, &response,
                                    &new_cookie);

        if (ret != RSI_SUCCESS) {
            return ret;
        }

        if (response == RSI_REJECT) {
            return RSI_ERROR_INPUT;
        }

        base = new_base;
        cookie = new_cookie;
    }

    return RSI_SUCCESS;
}
```

I₀₃₅₅ The Host programming model for handling an S2AP change request is to call RMI_RTT_SET_S2AP in a loop until progress reaches the top of the target IPA range, as shown in the following pseudocode:

```
int host_set_s2ap(unsigned long base, unsigned long top)
{
    unsigned long out_top, index, rtt_tree;
    int ret = RMI_SUCCESS;

    while (base != top) {
        ret = rmi_rtt_set_s2ap(rd, rec, base, top,
                              &out_top, &rtt_tree, &index);

        if (ret == RMI_ERROR_RTT || ret == RMI_ERROR_AUX_RTT) {
            create_rtt(ipa = out_top, level = index + 1, rtt_tree);
            continue;
        } else if (ret != RMI_SUCCESS) {
            break;
        }

        base = out_top;
    }

    return ret;
}
```

R₀₃₅₆ On REC entry following a REC exit due to S2AP change, `rec.s2ap_response` is set to the value of `enter.flags.s2ap_response`.

I₀₃₅₇

If all of the following are true then the output value of RSI_MEM_SET_PERM_INDEX indicates “Host rejected the request”:

- `rec.s2ap_addr` is not equal to `rec.s2ap_top`.
- `rec.s2ap_response` is REJECT.

Otherwise, the output value of RSI_MEM_SET_PERM_INDEX indicates “Host accepted the request”.

See also:

- [A4.3.11 REC exit due to S2AP change pending](#)
- [B3.72 RecS2APResponseToRsi function](#)
- [B4.3.30 RMI_REC_ENTER command](#)
- [B4.3.46 RMI_RTT_SET_S2AP command](#)
- [B5.3.10 RSI_MEM_SET_PERM_INDEX command](#)
- [D1.5.4 S2AP change flow](#)

A10.3.3.3 Stage 2 Access Permissions reset due to Host action

I₀₃₅₈

Execution of RMI_RTT_DESTROY or RMI_RTT_AUX_DESTROY sets the S2AP overlay index of the target RTTE to 0.

X₀₃₅₉

The Host is permitted at any time to destroy an RTT. Destruction of an RTT causes the RMM to lose information about the S2AP for the IPA range described by that RTT. The S2AP are observable by the Realm only while the RIPAS is RAM. Therefore, this specification defines the value to which the S2AP overlay index must be reset on a RIPAS transition to RAM.

See also:

- [A5.4 RIPAS change](#)

A10.4 Planes interrupts

Provisional

Decide whether this section should be integrated into the main [Realm interrupts](#) section.

- I₀₃₆₀ On REC creation, the GIC owner for the REC is P0.
- I₀₃₆₁ On Plane entry, P0 can transfer GIC ownership to the target Pn.
- I₀₃₆₂ On Plane entry, if P0 transfer GIC ownership to the target Pn then the GIC state in RsiPlaneEnter is ignored. This means that the GIC state of the owner is preserved and is shared across GIC ownership changes between planes.
- S₀₃₆₃ Allowing P0 to control which Plane is the GIC owner supports software usage models including the following:
- P0 is the GIC owner, and P0 emulates a vGIC for Pn, similar to how the Host emulates a vGIC for the Realm.
 - P0 is a lightweight “Pn switcher”, which does not emulate a vGIC. GIC ownership is transferred to the Pn which contains the main Realm guest OS.
- I₀₃₆₄ P0 can read the value of ICH_VTR_EL2 using the RSI_REALM_CONFIG command.
- I₀₃₆₅ On Plane exit, P0 is the GIC owner. If GIC ownership was transferred to Pn on Plane entry, it returns to P0 on Plane exit.
- I₀₃₆₆ On Plane exit, Pn’s GIC state is exposed to P0 via the RsiPlaneExit object.
- R₀₃₆₇ On REC entry the GIC state provided by the Host is assigned to the GIC owner.
- R₀₃₆₈ On REC entry, if the values of `enter.gicv3_lrs` describe one or more Pending interrupts and the most recent REC exit was from a Plane which is not the GIC owner then control returns to P0. This results in a Plane exit due to synchronous exception.
- R₀₃₆₉ On REC entry, if all of the following is true then control returns to P0, resulting in a Plane exit due to synchronous exception:
- The most recent REC exit was from Pn
 - The most recent REC exit was not from the GIC owner
 - The value of ICH_MISR_EL2 at the time of the REC exit was not zero.
- I₀₃₇₀ On REC exit, the Realm GIC state of the GIC owner Plane is reported to the Host.
- See also:
- [A2.3.2 REC attributes](#)
 - [A4.1 Realm exception model overview](#)
 - [A6.1 Realm interrupts](#)
 - [A10.2.3.1 Plane exit due to synchronous exception](#)
 - [B5.3.12 RSI_PLANE_ENTER command](#)
 - [B5.3.25 RSI_REALM_CONFIG command](#)

A10.5 Planes timers

Provisional

Decide whether this section should be integrated into the main [Realm timers](#) section.

- R₀₃₇₁ On REC exit from P0, the Realm EL1 timer state reported to the Host is P0's EL1 timer state.
- D₀₃₇₂ A Realm EL1 timer is *active* if it is enabled and unmasked.
- R₀₃₇₃ On REC exit from Pn, for each of the EL1 virtual and physical timers, if any of the following is true then the timer state reported to the Host is Pn's EL1 timer state:
- The Pn timer is active and the P0 timer is not active.
 - Both Pn and P0 timers are active and the Pn timer deadline is earlier than the P0 timer deadline.
- Otherwise, the timer state reported to the Host is P0's EL1 timer state.
- I₀₃₇₄ The following table summarises the timer state which is reported to the Host on REC exit.

P0 active	Pn active	Earliest CVAL	Reported to Host
0	0	P0	P0
0	0	Pn	P0
0	1	P0	Pn
0	1	Pn	Pn
1	0	P0	P0
1	0	Pn	P0
1	1	P0	P0
1	1	Pn	Pn

- I₀₃₇₅ On Plane exit, Pn's EL1 timer state is exposed to P0 via the RsiPlaneExit object.
- S₀₃₇₆ P0 software should check the Realm EL1 timer state on every return from RSI_PLANE_ENTER and update virtual interrupt state accordingly. This is true regardless of the value of `exit.exit_reason`: even if the return occurred for a reason unrelated to timers (for example, a Plane exit due to Data Abort), the Realm EL1 timer state should be checked.
- I₀₃₇₇ On Plane entry, the RMM may mask the hardware timer signal, following the same logic as for REC entry.
- U₀₃₇₈ Management of EL1 timer state for a Realm with multiple Planes can be implemented by multiplexing the following into the EL2 hardware timers:
- P0's EL1 timers
 - The Host's EL2 timers
- See also:
- [A6.2 Realm timers](#)
 - [A10.2.3 Plane exit](#)
 - [B5.4.17 RsiPlaneExit type](#)

Chapter A11

Realm memory encryption

This section describes encryption of physical memory which is accessible via Realm PAS. This encryption is transparent to Realm software, but has an impact on the security posture of a Realm.

- D₀₃₇₉ A Memory Encryption Context (MEC) is an encryption regime used to protect the memory owned by a Realm.
The memory protected by a Realm's MEC includes all memory which can be accessed by the Realm, and the RTTs which are owned by the Realm.
- U₀₃₈₀ Other Granules owned by the Realm, such as REC and RD, are protected with the RMM's MEC.
- D₀₃₈₁ A Memory Encryption Context Identifier (MECID) is a handle which is used to identify a MEC.
- I₀₃₈₂ The highest MECID value which the Host is permitted to pass via an RMI command is reported by the RMI_FEATURES command in RmiFeatureRegister1::MAX_MECID.
- D₀₃₈₃ A valid MECID is a MECID in the range $[0, \text{MAX_MECID}]$.
- R₀₃₈₄ On a platform which does not implement FEAT_MEC, MAX_MECID is zero.
- U₀₃₈₅ On a platform which implements FEAT_MEC, MAX_MECID is expected to be computed as follows:
- Determine the minimum MECID width supported across all system components capable of initiating Realm PAS transactions.
 - Determine the number of MECIDs which the platform needs to reserve for its own use. This is expected to be at least one, for protection of the RMM's memory.
 - Return $\text{MAX_MECID} = (2^{\text{MECID_WIDTH}}) - (\text{NUM_RESERVED_MECIDS} + 1)$

D₀₃₈₆ A MEC has a MEC state.

The MEC state values are shown in the following table.

Name	Description
MEC_STATE_PRIVATE_ASSIGNED	A Private MEC which is assigned to a Realm.
MEC_STATE_PRIVATE_UNASSIGNED	A Private MEC which is not assigned to a Realm.
MEC_STATE_SHARED	A Shared MEC.

- R₀₃₈₇ If MAX_MECID is zero then at platform boot, the state of MEC zero is MEC_STATE_SHARED.
- R₀₃₈₈ If MAX_MECID is not zero then at platform boot, the state of every MEC in the range $[0, \text{MAX_MECID}]$ is MEC_STATE_PRIVATE_UNASSIGNED.
- D₀₃₈₉ A Shared MEC has a set of zero or more member Realms.
- I₀₃₉₀ The input values of RMI_REALM_CREATE include a MECID.
- I₀₃₉₁ RMI_REALM_CREATE fails if any of the following is true:
- The MECID is not valid.
 - The state of the MEC is MEC_STATE_PRIVATE_ASSIGNED.
- I₀₃₉₂ On successful execution of RMI_REALM_CREATE:
- If the state of the MEC is MEC_STATE_PRIVATE_UNASSIGNED then the state becomes MEC_STATE_PRIVATE_ASSIGNED.
 - If the state of the MEC is MEC_STATE_SHARED then the new Realm is added to the Shared MEC.
- I₀₃₉₃ The RMI_MEC_SET_SHARED command changes the state of a MEC from MEC_STATE_PRIVATE_UNASSIGNED to MEC_STATE_SHARED.
- I₀₃₉₄ Execution of RMI_MEC_SET_SHARED fails if any of the following is true:
- The MECID provided by the Host is not valid.
 - The MECID provided by the Host identifies a MEC whose current state is not MEC_STATE_PRIVATE_UNASSIGNED.
 - Any valid MECID identifies a Shared MEC. This means that there can be at most a single Shared MEC in existence at a time.

- I₀₃₉₅ The RMI_MEC_SET_PRIVATE command changes the state of a MEC from MEC_STATE_SHARED to MEC_STATE_PRIVATE_UNASSIGNED.
- I₀₃₉₆ Execution of RMI_MEC_SET_PRIVATE fails if any of the following is true:
- MAX_MECID is zero.
 - The MECID provided by the Host is not valid.
 - The MECID provided by the Host identifies a MEC whose current state is not MEC_STATE_SHARED.
 - The MECID provided by the Host identifies a Shared MEC which contains a non-zero number of Realms.
- I₀₃₉₇ On a platform which reports MAX_MECID to be zero, all Realms use the Shared MEC identified by MECID zero.
- I₀₃₉₈ On a platform which reports MAX_MECID to be non-zero, the Host can choose between the following approaches:
- Use the Shared MEC for all Realms.
 - Assign a Private MEC to each Realm, with the total number of Realms not exceeding MAX_MECID.
 - Use the Shared MEC for some Realms; for the remaining Realms, assign a Private MEC to each, with the total number of this latter set of Realms not exceeding MAX_MECID.
- R₀₃₉₉ The Realm attestation token includes a claim which describes the *MEC policy* of the Realm. The MEC policy is derived from the MEC state as follows:
- If the MEC state is MEC_STATE_PRIVATE_ASSIGNED then the MEC policy is MEC_POLICY_PRIVATE.
 - If the MEC state is MEC_STATE_SHARED then the MEC policy is MEC_POLICY_SHARED.
- I₀₄₀₀ On platform boot, the encryption context associated with every MEC changes.
- I₀₄₀₁ When the state of a MEC changes, the encryption context associated with that MEC changes.
- I₀₄₀₂ To illustrate the points at which encryption contexts must change, consider the following sequence:

Step	Reason for encryption context change
Platform boot	Platform boot
RMI_REALM_CREATE(rd=a, mecid=0)	
RMI_REALM_DESTROY(rd_a)	MEC_STATE_PRIVATE_ASSIGNED to MEC_STATE_PRIVATE_UNASSIGNED
RMI_REALM_CREATE(rd=b, mecid=1)	
RMI_REALM_DESTROY(rd_b)	MEC_STATE_PRIVATE_ASSIGNED to MEC_STATE_PRIVATE_UNASSIGNED
RMI_MEC_SET_SHARED(mecid=0)	
RMI_REALM_CREATE(rd=c, mecid=0)	
RMI_REALM_CREATE(rd=d, mecid=0)	
RMI_REALM_DESTROY(rd_d)	
RMI_REALM_DESTROY(rd_c)	
RMI_MEC_SET_PRIVATE(mecid=0)	MEC_STATE_SHARED to MEC_STATE_PRIVATE_UNASSIGNED
RMI_REALM_CREATE(rd=e, mecid=0)	

See also:

- [Arm Architecture Reference Manual Supplement, The Realm Management Extension \(RME\), for Armv9-A \[2\]](#)
- [A3.15 Support for Realm memory encryption](#)

- [A7.2.3.1.7 Realm MEC policy claim](#)
- [B4.3.9 RMI_MEC_SET_PRIVATE command](#)
- [B4.3.10 RMI_MEC_SET_SHARED command](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [B4.3.26 RMI_REALM_DESTROY command](#)
- [B4.4.15 RmiFeatureRegister1 type](#)

DRAFT

DRAFT

Part B

Interface

Chapter B1

Commands

This chapter describes how RMM commands are defined in this specification.

B1.1 Overview

R _{VZRKZ}	<p>The RMM exposes the following interfaces to the Host:</p> <ul style="list-style-type: none"> • The <i>Realm Management Interface</i> (RMI)
R _{NPLKX}	<p>The RMM exposes the following interfaces to a Realm:</p> <ul style="list-style-type: none"> • The <i>Realm Services Interface</i> (RSI) • The <i>Power State Coordination Interface</i> (PSCI) <p>Any other SMC executed by a Realm returns SMCCC_NOT_SUPPORTED.</p>
I _{TKQXF}	An RMM interface consists of a set of RMM commands.
I _{RTRYT}	An RMM interface is compliant with the SMC Calling Convention (SMCCC).
R _{NNFPH}	SMCCC version ≥ 1.2 is required.
X _{FDXJG}	SMCCC version 1.2 increases the number of SMC64 arguments and return values from 4 to 17. Some RMM commands use more than 4 input or output values.
R _{VXJJQ}	On a CCA platform which implements FEAT_SVE, SMCCC version ≥ 1.3 is required.
X _{KCMSY}	SMCCC version 1.3 introduces a bit in the FID which a caller can use to indicate that SVE state does not need to be preserved across the SMC call.
R _{JNVJQ}	On a CCA platform which implements FEAT_SME, SMCCC version ≥ 1.4 is required.
X _{QXMZL}	SMCCC version 1.4 adds support for preservation of SME state across an SMC call.
R _{KWMVX}	An RMM command uses the SMC64 calling convention.
S _{DFNMZ}	<p>To determine whether an RMM interface is implemented, software should use the following flow:</p> <ol style="list-style-type: none"> 1. Determine whether the SMCCC_VERSION command is implemented, following the procedure described in Arm SMC Calling Convention [17]. 2. Check that the SMCCC version is ≥ 1.1. 3. Execute the <Interface>.Version command, which returns: <ul style="list-style-type: none"> • SMCCC_NOT_SUPPORTED (-1) if <Interface> is not implemented. • A version number (>0) if <Interface> is implemented.
R _{YBXKR}	<p>All data types defined in this specification are little-endian.</p> <p>See also:</p> <ul style="list-style-type: none"> • Chapter B4 Realm Management Interface • Chapter B5 Realm Services Interface • Chapter B6 Power State Control Interface

B1.2 Command definition

I_{WBMVP}

The definition of an RMM command consists of:

- A *function identifier* (FID)
- A set of *input values* (referred to as “arguments” in SMCCC)
- A set of *output values* (referred to as “results” in SMCCC)
- A set of *context values*
- A partially-ordered set of *failure conditions*
- A set of *success conditions*
- A set of *footprint items*

I_{GCVWC}

Each failure condition, success condition and footprint item has an associated identifier. Identifiers are unique within each of the above groups, within each command.

An identifier has no meaning. It is only a label by which a given condition or footprint item can be referred to.

R_{STJHR}

On calling an RMI or RSI command, any of X1 - X16 which are not specified as input values in the command definition SBZ.

R_{KBWJD}

On return from an RMI or RSI command, any of X0 - X16 which are not specified as output values in the command definition MBZ.

See also:

- SMCCC [Arm SMC Calling Convention](#) [17]

B1.2.1 Example command

I_{NFVGF}

The following command, EXAMPLE_ADD, is an example of how the components of an RMM command definition are presented in this document.

This command takes as an input value the address `params_ptr` of an NS Granule which contains two integer values `x` and `y`. On successful execution of the command:

- The output value `sum` contains the sum of `x` and `y`
- The output value `zero` indicates whether either of `x` or `y` is zero

EXAMPLE_ADD is defined as follows:

Interface

FID

0x042

Input values

Name	Register	Field	Type	Description
<code>fid</code>	X0	[63:0]	UInt64	Command FID
<code>params_ptr</code>	X1	[63:0]	Address	PA of parameters

Context

The EXAMPLE_ADD command operates on the following context.

Name	Type	Value	Before	Description
params	ExampleParams	Params(params_ptr)	false	Parameters

Output values

Name	Register	Field	Type	Description
result	x0	[15:0]	CommandReturnCode	Command return status
sum	x1	[63:0]	UInt64	Sum of x and y
zero	x2	[63:0]	UInt64	Whether either x or y was zero

Failure conditions

ID	Condition
params_align	pre: !AddrIsGranuleAligned(params_ptr) post: ResultEqual(result, ERROR_INPUT)
params_gpt	pre: Granule(params_ptr).gpt != GPT_NS post: ResultEqual(result, ERROR_MEMORY)

Success conditions

ID	Post-condition
sum	sum == params.x + params.y
zero	zero == (params.x == 0) (params.y == 0)

B1.3 Command registers

D _{ZDGNM}	An <i>FID</i> is a value which identifies a particular RMM command.
I _{MJQ GK}	The FID of an RMM command is unique among the RMM commands in an RMM interface.
I _{RVPGY}	An FID is read from general-purpose register X0.
D _{XL SFS}	An <i>input value</i> is a value read by an RMM command from general-purpose registers.
D _{VCDCW}	An <i>output value</i> is a value written by an RMM command to general-purpose registers.
D _{CZLVJ}	A <i>command return code</i> is a value which specifies whether an RMM command succeeded or failed.
I _{FRZFT}	A command return code is written to general-purpose register X0.

B1.4 Command condition expressions

D _{CHRYB}	A <i>condition expression</i> is an expression which evaluates to a boolean value.
--------------------	--

I_{BPNKQ} Following expansion of macros, a *condition expression* is a valid expression in Arm Specification Language (ASL).
See also:

- [Arm Specification Language Reference Manual \[18\]](#)
- [Chapter B3 Command condition functions](#)

B1.5 Command context values

D_{DLBYC} A *context value* is a value which is derived from the value of a command input register or command output register and which is used by a command condition expression.

I_{VKKKY} A context value can be thought of as a local variable for use by command condition expressions.

For example, consider the following example command condition expression:

```
!AddrIsGranuleAligned(RealmParams(params_ptr).rtt_base)
```

By introducing a context value `params` with the value `RealmParams(params_ptr)`, this command condition expression can be re-written as:

```
!AddrIsGranuleAligned(params.rtt_base)
```

D_{QDFNW} The *before* property of a context value indicates whether its expression is re-evaluated after the command has executed.

- `before = true`: the expression is not re-evaluated after the command has executed
- `before = false`: the expression is re-evaluated after the command has executed

I_{LTLQN} Specifying `before = true` for a context value allows system state to be sampled before command execution, and then used after command execution in a command success condition.

For example, the `RMI_REALM_DESTROY` command takes as an input value the address `rd` of a Realm Descriptor. Successful execution of the command results observable effects including the following:

- The state of the RD Granule changes from `RD` to `DELEGATED`
- The state of the RTT base Granule, whose address was previously held in the RD, changes from `RTT` to `DELEGATED`

The address of the RTT base Granule is not included in the input values of the command.

A context value is defined as follows:

Name	Type	Value	Before	Description
<code>rtt_base</code>	Address	<code>Realm(rd).rtt_base</code>	<code>true</code>	RTT base address

The state change of the RTT Granule can then be expressed as:

```
Granule(rtt_base).state == DELEGATED
```

I_{YNDGD} The *before* property of a context value has no effect if the value is only used in command failure conditions.

D_{XBHPB} An *in-memory value* is a value passed to a command via an in-memory data structure, the address of which is passed in an input register.

I_{ZTYSS} An in-memory value is a context value.

See also:

- [B4.3.25 RMI_REALM_CREATE command](#)

B1.6 Command failure conditions

D_{DNQOC} An RMM command *failure condition* defines a way in which the command can fail.

I_{GVBBZ} A failure condition consists of a *pre-condition* and a *post-condition*.

I_{WTSZH} A failure pre-condition can be thought of as the “trigger” of the failure: if the pre-condition is true then the command fails.

I_{KJHNX} A failure post-condition can be thought of as the “effect” of the failure: if the command failed due to a particular trigger, then the post-condition defines the error code which is returned.

I_{CVTGY} A failure pre-condition is a condition expression whose terms can include input values and context values.

I_{HNDNN} A failure post-condition is a condition expression whose terms can include input values and context values.

I_{KHJDY} Observability of the checking of command failure conditions is subject to a partial order.

An ordering relation “*A* precedes *B*” means either of the following:

- The pre-condition of *B* is well-formed only if the pre-condition of *A* is false. This is referred to as a *well-formedness ordering*.
- If the pre-conditions of *A* and *B* are both true, then the post-condition of *A* is observed. This is referred to as a *behavioral ordering*.

The absence of an ordering relation “*A* precedes *B*” means that, if the pre-conditions of *A* and *B* are both true then either the post-condition of *A* is observed or the post-condition of *B* is observed.

Orderings are specified between groups of failure conditions. For example, the expression $[A, B] < [C, D]$ means that both conditions *A* and *B* precede both conditions *C* and *D*.

The same information is also presented graphically, with failure conditions represented as nodes and ordering relations represented as edges.

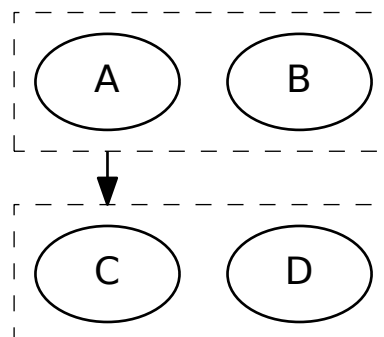


Figure B1.1

The specification does not state whether an individual ordering relation is a well-formedness ordering or a behavioral ordering.

I _{JMTTY}	A given implementation of the RMM is expected to have deterministic behavior. That is, for a runtime instance of the RMM in a particular state, two executions of a command without an interleaving of other commands, with the same input values, results in the same outcome (either success, or the same failure condition.)
R _{WXZJJ}	If a failure pre-condition evaluates to true then the corresponding failure post-condition evaluates to true.
R _{DDGDW}	If a failure pre-condition evaluates to true then the command is aborted.
R _{TFZMS}	If a command fails then all output values except for X0 are UNDEFINED, unless stated otherwise.
R _{VHFHD}	If no failure pre-condition evaluates to true then the command succeeds.

B1.7 Command success conditions

D _{SZGNZ}	An RMM command <i>success condition</i> defines an observable effect of a successful execution of the command.
I _{LZXHB}	A success condition is a condition expression whose terms can include input values, context values and output values.
I _{NMCSF}	The order in which success conditions are listed has no architectural significance.
I _{NJQFG}	If an RMM command succeeds then the return code is <Interface>_SUCCESS.
R _{MKRJV}	If an RMM command succeeds then all of its success conditions evaluate to true.

B1.8 Concrete and abstract types

D _{NXQWV}	<p>A <i>concrete type</i> is a type which has a defined encoding.</p> <p>Examples of concrete types include:</p> <ul style="list-style-type: none"> • An integer which has a defined bit width. • An enumeration within which each label is associated with a unique binary value. • A struct which has a defined width, and within which each member has a defined position. The type of each member of a concrete struct is a concrete type.
I _{WDGMW}	Concrete types are used to define command input values and output values.
D _{WTCVJ}	<p>An <i>abstract type</i> is a type which does not have a defined encoding.</p> <p>Examples of concrete types include:</p> <ul style="list-style-type: none"> • An integer which does not have a defined bit width. • An enumeration which has a set of labels, but which does not define a binary value for each label. • A struct which has a set of members, but which does not define a struct width nor a position for each member. <p>The type of each member of an abstract struct is an abstract type.</p>
I _{QZRGY}	Abstract types are used to model the internal state of the RMM.
I _{LMKGP}	<p>A command failure condition or success condition may need to test for logical equality between a concrete type and a corresponding abstract type. For example, the command may set the value of an internal RMM variable to match the value of a command input. To enable such comparisons, the specification defines an <code>Equal()</code> function for each pair of corresponding concrete and abstract types.</p> <p>See also:</p> <ul style="list-style-type: none"> • B3.26 Equal function

B1.9 Command footprint

D _{ZDJDB}	The <i>footprint</i> of an RMM command defines the set of state items which successful execution of the command can modify.
I _{XMZYS}	The footprint of an RMM command may include state items which are not modified by successful execution of the command.
I _{RWQMJ}	If an RMM command changes the state of a Granule then the footprint typically does not include all attributes of the object which is created or destroyed. For example, the footprint of RMI_REALM_CREATE includes the state of the RD Granule, but does not include attributes of the newly-created Realm.
R _{WZYBV}	Except for items in the footprint of an RMM command and registers in the output values of the RMM command, execution of the command does not have any observable effects.

B1.10 Command testing

I _{MBNZM}	Command definitions can be used to generate testbenches which check whether an implementation complies with the specified failure and success conditions.
--------------------	---

I _{JGGJN}	A testbench for the EXAMPLE_ADD command presented above would look similar to the following:
--------------------	--

```
// Test EXAMPLE_ADD command
Test_ExampleAdd(Registers regs_in)

    // Unpack input values
    RmmPa params_ptr = regs_in.X1;

    // Evaluate context values
    ExampleParams params = ExampleParams(params_ptr);

    // Evaluate failure pre-conditions
    boolean params_align_pre = !AddrIsGranuleAligned(params_ptr);
    boolean params_gpt_pre = Granule(params_ptr).gpt != GPT_NS;

    // Execute command
    regs_out = RmiExampleAdd(regs_in);

    // Pack output values
    CommandReturnCode result = regs_out.X0;
    integer sum = regs_out.X1;
    integer zero = regs_out.X2;

    // Check return code
    boolean success = (result == Success);

    // Evaluate failure post-conditions
    boolean params_align_post = result == Status(ErrorInput, 1);
    boolean params_gpt_post = result == Status(ErrorInputMemory, 0);

    // Evaluate success conditions
    boolean sum_post = sum == params.x + params.y;
    boolean zero_post = zero == (params.x == 0) || (params.z == 0);

    // Check failure conditions, in order specified
    assert params_align_pre IMPLIES params_align_post;
    assert (!params_align_pre && params_gpt_pre) IMPLIES params_gpt_post;

    // Check that, if no failure pre-condition was violated, command succeeded
    assert (!params_align_pre && !params_gpt_pre) IMPLIES success;
```



```
// Check success conditions, without any ordering
assert success IMPLIES S01_post;
assert success IMPLIES S02_post;
```

Note that the syntax `x IMPLIES y`, which is logically equivalent to `!x || y`, is not yet defined in ASL.

DRAFT

Chapter B2

Interface versioning

This section describes how the RMI and RSI interfaces are versioned, and how the caller of each can determine whether there exists a mutually acceptable revision of the interface via which it can communicate with the RMM.

Other interfaces exposed by the RMM, such as PSCI, may define their own versioning schemes which differ from that used by RMI and RSI. For details, refer to the specification of the interface concerned.

\mathcal{I}_{LZVQR}

Revisions of the RMI and the RSI are identified by a (major, minor) version tuple.

The semantics of this version tuple are as follows. For two revisions of the interface $P = (maj_P, min_P)$ and $Q = (maj_Q, min_Q)$:

- If $maj_P \neq maj_Q$ then the two interfaces may contain incompatible commands.
- If $maj_P == maj_Q$ and $min_P < min_Q$ then:
 - Every command defined in P has the same behavior in Q, when called with input values that are specified as valid in P.
 - A command defined in P may accept additional input values in Q. These could be provided via any of:
 - * Input registers which were unused in P.
 - * Input memory locations which were specified as SBZ in P.
 - * Encodings which were specified as reserved in P.
 - A command defined in P may return additional output values in Q. These could be returned via any of:
 - * Output registers which were unused in P.
 - * Output memory locations which were specified as MBZ in P.
 - * Encodings which were specified as reserved in P.
 - Q may contain additional commands which are not present in P.
- P is *less than* Q if one of the following conditions is true:
 - $maj_P < maj_Q$
 - $maj_P == maj_Q$ and $min_P < min_Q$

 \mathcal{I}_{ZCPBC}

For each interface, an RMM implementation supports a set of revisions. The size of this set is at least one.

 \mathcal{I}_{RMSLZ}

If an RMM implementation supports a given interface revision (x, y) then Arm expects that it will also supports all earlier revisions with the same major version number. That is:

$(x, 0), (x, 1) \dots (x, y-1), (x, y)$.

A possible exception to this may occur if a security vulnerability is discovered in a particular revision of the interface. For example, if interface revision (x, bad) is found to contain a vulnerability then an RMM implementation may choose to support the following set of revisions:

$(x, 0), (x, 1) \dots (x, bad-1), (x, bad+1) \dots (x, y-1), (x, y)$.

 \mathcal{I}_{GLDQG}

The set of interface revisions supported by an RMM implementation may include revisions with different major version numbers, for example:

$(1, 0), (1, 1) \dots (1, m)$

$(2, 0), (2, 1) \dots (2, n)$

 \mathcal{I}_{JNVXJ}

The RMI_VERSION and RSI_VERSION commands allow the caller and the RMM to determine whether there exists a mutually acceptable revision of the interface via which the two components can communicate.

In each case:

- The caller provides a requested interface revision.
- The output values include a status code and two revisions which are supported by the RMM: a *lower revision* and a *higher revision*.
- The *higher revision* value is the highest interface revision which is supported by the RMM.
- The *lower revision* is less than or equal to the *higher revision*.

The status code and *lower revision* output values indicate which of the following is true, in order of precedence:

- a) The RMM supports an interface revision which is compatible with the requested revision.
 - The status code is “success”.

- The *lower revision* is equal to the requested revision.
- b) The RMM does not support an interface revision which is compatible with the requested revision. The RMM supports an interface revision which is incompatible with and less than the requested revision.
 - The status code is “failure”.
 - The *lower revision* is the highest interface revision which is both less than the requested revision and supported by the RMM.
- c) The RMM does not support an interface revision which is compatible with the requested revision. The RMM supports an interface revision which is incompatible with and greater than the requested revision.
 - The status code is “failure”.
 - The *lower revision* is equal to the *higher revision*.

The following table shows how each of a set of example scenarios maps onto the above outcomes.

Scenario	Revisions supported by RMM	Revision requested by caller	Outcome	“Lower revision” output value	“Higher revision” output value
1	(1, 0)	(1, 0)	Success (a)	(1, 0)	(1, 0)
2	(1, 0), (1, 1)	(1, 0)	Success (a)	(1, 0)	(1, 1)
3	(1, 0), (2, 0)	(1, 0)	Success (a)	(1, 0)	(2, 0)
4	(1, 0)	(1, 1)	Failure (b)	(1, 0)	(1, 0)
5	(1, 0), (1, 1)	(1, 2)	Failure (b)	(1, 1)	(1, 1)
6	(1, 0), (1, 1)	(2, 0)	Failure (b)	(1, 1)	(1, 1)
7	(1, 0), (1, 1), (1, 3)	(1, 2)	Failure (b)	(1, 1)	(1, 3)
8	(1, 0)	(2, 0)	Failure (b)	(1, 0)	(1, 0)
9	(1, 0)	(2, 1)	Failure (b)	(1, 0)	(1, 0)
10	(1, 0), (1, 1)	(2, 0)	Failure (b)	(1, 1)	(1, 1)
11	(1, 0), (1, 1)	(2, 1)	Failure (b)	(1, 1)	(1, 1)
12	(1, 0), (1, 1), (2, 0)	(2, 1)	Failure (b)	(2, 0)	(2, 0)
13	(2, 0)	(1, 0)	Failure (c)	(2, 0)	(2, 0)
14	(2, 0)	(1, 1)	Failure (c)	(2, 0)	(2, 0)
15	(2, 0), (2, 1)	(1, 0)	Failure (c)	(2, 1)	(2, 1)

See also:

- [B4.1 RMI version](#)
- [B4.3.56 RMI_VERSION command](#)
- [B5.1 RSI version](#)
- [B5.3.26 RSI_VERSION command](#)

Chapter B3

Command condition functions

This chapter describes functions which are used in command condition expressions.

See also:

- [B1.4 Command condition expressions](#)

B3.1 AddrInRange function

Returns TRUE if `addr` is within `[base, base+size]`.

```
func AddrInRange(  
    addr : Address,  
    base : Address,  
    size : integer) => boolean  
begin  
    return ((UInt(addr) >= UInt(base))  
        && (UInt(addr) <= UInt(base) + size));  
end
```

B3.2 AddrIsAligned function

Returns TRUE if address `addr` is aligned to an `n` byte boundary.

```
func AddrIsAligned(  
    addr : Address,  
    n : integer) => boolean
```

B3.3 AddrIsAuxLive function

Returns TRUE if IPA `addr` is *auxiliary-live*, that is live in any auxiliary RTT.

```
func AddrIsAuxLive(  
    addr : Address,  
    realm : RmmRealm) => boolean
```

B3.4 AddrIsGranuleAligned function

Returns TRUE if address `addr` is aligned to the size of a Granule.

```
func AddrIsGranuleAligned(  
    addr : Address) => boolean
```

```
func AddrIsGranuleAligned(  
    addr : integer) => boolean
```

See also:

- [A2.2 Granule](#)

B3.5 AddrIsProtected function

Returns TRUE if address `addr` is a Protected IPA for `realm`.

```
func AddrIsProtected(  
    addr : Address,  
    realm : RmmRealm) => boolean  
begin  
    return UInt(addr) < 2^(realm.ipa_width - 1);  
end
```

B3.6 AddrIsRttLevelAligned function

Returns TRUE if Address `addr` is aligned to the size of the address range described by an RTTE in a level `level` RTT.

Returns FALSE if `level` is invalid.

```
func AddrIsRttLevelAligned(  
    addr : Address,  
    level : integer) => boolean
```

B3.7 AddrIsWithin function

Returns TRUE if address `addr` is within the outer range [`base`, `top`).

```
func AddrIsWithin(  
    addr : Address,  
    base : Address,  
    top : Address) => boolean  
begin  
    var addr_int : integer = UInt(addr);  
    var top_int : integer = UInt(top);  
    var base_int : integer = UInt(base);  
    return ((UInt(addr) >= UInt(base))  
        && (UInt(addr) < UInt(top)));  
end
```

B3.8 AddrRangelsAuxLive function

Returns TRUE if any IPA in range [base, top) is *auxiliary-live*, that is live in any auxiliary RTT.

```
func AddrRangeIsAuxLive(  
    base : Address,  
    top : Address,  
    realm : RmmRealm) => boolean
```

B3.9 AddrRangelsProtected function

Returns TRUE if all addresses in range [base, top) are Protected IPAs for realm.

```
func AddrRangeIsProtected(  
    base : Address,  
    top : Address,  
    realm : RmmRealm) => boolean  
begin  
    var size = UInt(top) - UInt(base);  
    return (AddrIsProtected(base, realm)  
        && size > 0  
        && size < 2^realm.ipa_width  
        && AddrIsProtected(ToAddress(UInt(top) - 1), realm));  
end
```

B3.10 AddrRangelsWithin function

Returns TRUE if all addresses in the inner range [inner_base, inner_top) are within the outer range [outer_base, outer_top).

```
func AddrRangeIsWithin(  
    inner_base : Address,  
    inner_top : Address,  
    outer_base : Address,  
    outer_top : Address) => boolean  
begin  
    return (AddrIsWithin(inner_base, outer_base, outer_top)  
        && AddrIsWithin(inner_top, outer_base, outer_top));  
end
```

B3.11 AlignDownToRttLevel function

Round down addr to align to the size of the address range described by an RTTE in a level level RTT.

```
func AlignDownToRttLevel(  
    addr : Address,  
    level : integer) => Address
```

B3.12 AlignUpToRttLevel function

Round up addr to align to the size of the address range described by an RTTE in a level level RTT.

```
func AlignUpToRttLevel(  
    addr : Address,  
    level : integer) => Address
```

B3.13 AuxAlias16 function

Returns TRUE if any of the first `count` entries in a list of auxiliary Granule addresses are aliased - either among themselves, or with the address of another RMM object.

```
func AuxAlias16(  
    obj : Address,  
    aux : array [16] of Address,  
    count : integer) => boolean  
begin  
    assert 0 <= count && count <= 16;  
    var sorted = AuxSort(aux, count);  
  
    for i = 0 to count - 1 do  
        if sorted[i] == obj then  
            return TRUE;  
        end  
        if i >= 1 && sorted[i] == sorted[i - 1] then  
            return TRUE;  
        end  
    end  
    return FALSE;  
end
```

B3.14 AuxAlias32 function

Returns TRUE if any of the first `count` entries in a list of auxiliary Granule addresses are aliased - either among themselves, or with the address of another RMM object.

```
func AuxAlias32(  
    obj : Address,  
    aux : array [32] of Address,  
    count : integer) => boolean  
begin  
    assert 0 <= count && count <= 32;  
    var sorted = AuxSort(aux, count);  
  
    for i = 0 to count - 1 do  
        if sorted[i] == obj then  
            return TRUE;  
        end  
        if i >= 1 && sorted[i] == sorted[i - 1] then  
            return TRUE;  
        end  
    end  
    return FALSE;  
end
```

B3.15 AuxAligned16 function

Returns TRUE if the first `count` entries in a list of auxiliary Granule addresses are aligned to the size of a Granule.

```
func AuxAligned16(  
    aux : array [16] of Address,  
    count : integer) => boolean  
begin  
    assert 0 <= count && count <= 16;  
    for i = 0 to count - 1 do  
        if !AddrIsGranuleAligned(aux[i]) then
```



```
        return FALSE;
    end
end
return TRUE;
end
```

B3.16 AuxAligned32 function

Returns TRUE if the first `count` entries in a list of auxiliary Granule addresses are aligned to the size of a Granule.

```
func AuxAligned32(
    aux : array [32] of Address,
    count : integer) => boolean
begin
    assert 0 <= count && count <= 32;
    for i = 0 to count - 1 do
        if !AddrIsGranuleAligned(aux[i]) then
            return FALSE;
        end
    end
    return TRUE;
end
```

B3.17 AuxEqual16 function

Returns TRUE if the first `count` entries in two lists of auxiliary Granule addresses are equal.

```
func AuxEqual16(
    aux1 : array [16] of Address,
    aux2 : array [16] of Address,
    count : integer) => boolean
begin
    assert 0 <= count && count <= 16;
    for i = 0 to count - 1 do
        if aux1[i] != aux2[i] then
            return FALSE;
        end
    end
    return TRUE;
end
```

B3.18 AuxEqual32 function

Returns TRUE if the first `count` entries in two lists of auxiliary Granule addresses are equal.

```
func AuxEqual32(
    aux1 : array [32] of Address,
    aux2 : array [32] of Address,
    count : integer) => boolean
begin
    assert 0 <= count && count <= 32;
    for i = 0 to count - 1 do
        if aux1[i] != aux2[i] then
            return FALSE;
        end
    end
    return TRUE;
end
```

B3.19 AuxSort function

Sort first `count` entries in array of auxiliary Granule addresses.

```
func AuxSort(  
    addrs : array [16] of Address,  
    count : integer) => array [16] of Address
```

```
func AuxSort(  
    addrs : array [32] of Address,  
    count : integer) => array [32] of Address
```

B3.20 AuxStateEqual16 function

Returns TRUE if the state of the first `count` entries in a list of auxiliary Granule addresses is equal to `state`.

```
func AuxStateEqual16(  
    aux : array [16] of Address,  
    count : integer,  
    state : RmmGranuleState) => boolean  
begin  
    assert 0 <= count && count <= 16;  
    for i = 0 to count - 1 do  
        if (!PaIsDelegable(aux[i])  
            || GranuleAt(aux[i]).state != state) then  
            return FALSE;  
        end  
    end  
    return TRUE;  
end
```

B3.21 AuxStateEqual32 function

Returns TRUE if the state of the first `count` entries in a list of auxiliary Granule addresses is equal to `state`.

```
func AuxStateEqual32(  
    aux : array [32] of Address,  
    count : integer,  
    state : RmmGranuleState) => boolean  
begin  
    assert 0 <= count && count <= 32;  
    for i = 0 to count - 1 do  
        if (!PaIsDelegable(aux[i])  
            || GranuleAt(aux[i]).state != state) then  
            return FALSE;  
        end  
    end  
    return TRUE;  
end
```

B3.22 AuxStates function

Inductive function which identifies the states of the first `count` entries in a list of auxiliary Granules.

This function is used in the definition of command footprint.

```
func AuxStates(  
    aux : array [16] of Address,  
    count : integer)
```

```
func AuxStates(  
    aux : array [32] of Address,  
    count : integer)
```

B3.23 CurrentRealm function

Returns the current Realm.

```
func CurrentRealm() => RmmRealm
```

B3.24 CurrentRec function

Returns the current REC.

```
func CurrentRec() => RmmRec
```

B3.25 DeviceCommunicate function

Process device communication data and return the new state of the device transaction.

```
func DeviceCommunicate(  
    pdev : RmmPdev,  
    data : RmiDevCommData) => RmmDevCommState
```

```
func DeviceCommunicate(  
    vdev : RmmVdev,  
    data : RmiDevCommData) => RmmDevCommState
```

```
func DeviceCommunicate(  
    vdev : RmmRdev) => RmmDevCommState
```

B3.26 Equal function

Check whether concrete and abstract values are equal

```
func Equal(  
    abstract : RmmFeature,  
    concrete : RmiFeature) => boolean
```

```
func Equal(  
    concrete : RmiFeature,  
    abstract : RmmFeature) => boolean
```

```
func Equal(  
    abstract : RmmHashAlgorithm,  
    concrete : RmiHashAlgorithm) => boolean
```

```
func Equal(  
    concrete : RmiHashAlgorithm,  
    abstract : RmmHashAlgorithm) => boolean
```

```
func Equal(  
    abstract : RmmLfaPolicy,  
    concrete : RmiLfaPolicy) => boolean
```

```
func Equal(  
    concrete : RmiLfaPolicy,  
    abstract : RmmLfaPolicy) => boolean
```

```
func Equal (
    abstract : RmmPdevProtection,
    concrete : RmiPdevProtection) => boolean
```

```
func Equal (
    concrete : RmiPdevProtection,
    abstract : RmmPdevProtection) => boolean
```

```
func Equal (
    abstract : RmmPdevState,
    concrete : RmiPdevState) => boolean
```

```
func Equal (
    concrete : RmiPdevState,
    abstract : RmmPdevState) => boolean
```

```
func Equal (
    abstract : RmmRecRunnable,
    concrete : RmiRecRunnable) => boolean
```

```
func Equal (
    concrete : RmiRecRunnable,
    abstract : RmmRecRunnable) => boolean
```

```
func Equal (
    abstract : RmmRipas,
    concrete : RmiRipas) => boolean
```

```
func Equal (
    concrete : RmiRipas,
    abstract : RmmRipas) => boolean
```

```
func Equal (
    abstract : RmmRttPlaneFeature,
    concrete : RmiRttPlaneFeature) => boolean
```

```
func Equal (
    concrete : RmiRttPlaneFeature,
    abstract : RmmRttPlaneFeature) => boolean
```

```
func Equal (
    abstract : RmmRttS2APEncoding,
    concrete : RmiRttS2APEncoding) => boolean
```

```
func Equal (
    concrete : RmiRttS2APEncoding,
    abstract : RmmRttS2APEncoding) => boolean
```

```
func Equal (
    abstract : RmmVdevState,
    concrete : RmiVdevState) => boolean
```

```
func Equal (
    concrete : RmiVdevState,
    abstract : RmmVdevState) => boolean
```

```
func Equal (
    abstract : RmmRdevState,
    concrete : RsiDeviceState) => boolean
```

```
func Equal (
```

```

        concrete : RsiDeviceState,
        abstract  : RmmRdevState) => boolean
    
```

```

func Equal (
    abstract : RmmFeature,
    concrete : RsiFeature) => boolean
    
```

```

func Equal (
    concrete : RsiFeature,
    abstract  : RmmFeature) => boolean
    
```

```

func Equal (
    abstract : RmmHashAlgorithm,
    concrete : RsiHashAlgorithm) => boolean
    
```

```

func Equal (
    concrete : RsiHashAlgorithm,
    abstract  : RmmHashAlgorithm) => boolean
    
```

```

func Equal (
    abstract : RmmRipas,
    concrete : RsiRipas) => boolean
    
```

```

func Equal (
    concrete : RsiRipas,
    abstract  : RmmRipas) => boolean
    
```

```

func Equal (
    abstract : RmmRipasChangeDestroyed,
    concrete : RsiRipasChangeDestroyed) => boolean
    
```

```

func Equal (
    concrete : RsiRipasChangeDestroyed,
    abstract  : RmmRipasChangeDestroyed) => boolean
    
```

See also:

- [B1.8 Concrete and abstract types](#)

B3.27 FeatureToRmi function

Convert feature bit to RMI type.

```

func FeatureToRmi (
    value : RmmFeature) => RmiFeature
begin
    case value of
        when FEATURE_FALSE => return RMI_FEATURE_FALSE;
        when FEATURE_TRUE  => return RMI_FEATURE_TRUE;
    end
end
    
```

B3.28 FeatureToRsi function

Convert feature bit to RSI type.

```

func FeatureToRsi (
    value : RmmFeature) => RsiFeature
begin
    case value of
    
```

```
        when FEATURE_FALSE => return RSI_FEATURE_FALSE;
        when FEATURE_TRUE  => return RSI_FEATURE_TRUE;
    end
end
```

B3.29 Gicv3ConfigIsValid function

Returns TRUE if the values of all gicv3_* attributes are valid.

```
func Gicv3ConfigIsValid(
    gicv3_hcr : bits(64),
    gicv3_lrs : array [16] of bits(64)) => boolean
```

See also:

- [A6.1 Realm interrupts](#)
- [B4.4.31 RmiRecEnter type](#)

B3.30 GranuleAccessPermitted function

Returns TRUE if the Granule located at physical address `addr` is accessible via `pas`.

```
func GranuleAccessPermitted(
    addr : Address,
    pas : RmmPhysicalAddressSpace) => boolean
begin
    case GranuleAt(addr).gpt of
        when GPT_NS      => return (pas == PAS_NS);
        when GPT_REALM   => return (pas == PAS_REALM);
        when GPT_SECURE  => return (pas == PAS_SECURE);
        when GPT_ROOT    => return (pas == PAS_ROOT);
        when GPT_AAP     => return TRUE;
    end
end
```

B3.31 GranuleAt function

Returns the Granule located at physical address `addr`.

```
func GranuleAt(
    addr : Address) => RmmGranule
```

See also:

- [A2.2 Granule](#)

B3.32 ImplFeatures function

Returns features supported by the implementation.

```
func ImplFeatures() => RmmFeatures
```

See also:

- [Chapter A3 Feature discovery and configuration](#)

B3.33 MecMembers function

Returns number of Realms which are members of a given MEC.

```
func MecMembers(  
    mecid : bits(64)) => integer
```

See also:

- [Chapter A11 Realm memory encryption](#)

B3.34 MecPolicy function

Returns policy associated with a given MEC.

```
func MecPolicy(  
    mecid : bits(64)) => RmmMecPolicy  
begin  
    case MecState(mecid) of  
        when MEC_STATE_SHARED           => return MEC_POLICY_SHARED;  
        when MEC_STATE_PRIVATE_ASSIGNED => return MEC_POLICY_PRIVATE;  
        when MEC_STATE_PRIVATE_UNASSIGNED => return MEC_POLICY_PRIVATE;  
    end  
end
```

See also:

- [Chapter A11 Realm memory encryption](#)

B3.35 MecState function

Returns state of a given MEC.

```
func MecState(  
    mecid : bits(64)) => RmmMecState
```

See also:

- [Chapter A11 Realm memory encryption](#)

B3.36 MemPermLabelSupported function

Returns TRUE if the specified value is a valid encoding for a memory permission label and the label is supported by the implementation.

```
func MemPermLabelSupported(  
    label : bits(64)) => boolean
```

B3.37 MinAddress function

Returns the smaller of two addresses.

```
func MinAddress(  
    addr1 : Address,  
    addr2 : Address) => Address  
begin  
    return ToAddress(Min(UInt(addr1), UInt(addr2)));  
end
```

B3.38 MpidrEqual function

Returns TRUE if the specified MPIDR values are logically equivalent.

```
func MpidrEqual(  
    rmm_mpidr : bits(64),  
    rmi_mpidr : RmiRecMpidr) => boolean  
begin  
    return (rmm_mpidr[ 3: 0] == rmi_mpidr.aff0  
        && rmm_mpidr[15: 8] == rmi_mpidr.aff1  
        && rmm_mpidr[23:16] == rmi_mpidr.aff2  
        && rmm_mpidr[31:24] == rmi_mpidr.aff3);  
end
```

B3.39 MpidrIsUsed function

Returns TRUE if the specified MPIDR value identifies a REC in the current Realm.

```
func MpidrIsUsed(  
    mpidr : bits(64)) => boolean
```

B3.40 MsiAddrIsValid function

Returns TRUE if addr is a valid MSI address.

```
func MsiAddrIsValid(  
    addr : Address) => boolean
```

B3.41 PalsDelegable function

Returns TRUE if the Granule located at physical address addr is Delegable memory.

```
func PaIsDelegable(  
    addr : Address) => boolean  
begin  
    return (PaIsDelegableDevMem(addr)  
        || PaIsDelegableDram(addr));  
end
```

B3.42 PalsDelegableCohDevMem function

Returns TRUE if the Granule located at physical address addr is Delegable coherent device memory.

```
func PaIsDelegableCohDevMem(  
    addr : Address) => boolean
```

B3.43 PalsDelegableDevMem function

Returns TRUE if the Granule located at physical address addr is Delegable device memory.

```
func PaIsDelegableDevMem(  
    addr : Address) => boolean  
begin  
    return (PaIsDelegableCohDevMem(addr)  
        || PaIsDelegableNonCohDevMem(addr));  
end
```

B3.44 PalsDelegableDram function

Returns TRUE if the Granule located at physical address `addr` is Delegable DRAM.

```
func PaIsDelegableDram(  
    addr : Address) => boolean
```

B3.45 PalsDelegableNonCohDevMem function

Returns TRUE if the Granule located at physical address `addr` is Delegable non-coherent device memory.

```
func PaIsDelegableNonCohDevMem(  
    addr : Address) => boolean
```

B3.46 PdevAt function

Returns the PDEV object located at physical address `addr`.

```
func PdevAt (  
    addr : Address) => RmmPdev
```

B3.47 PdevAuxCount function

Returns the number of auxiliary Granules required for a PDEV with the specified flags.

The return value is guaranteed not to be greater than 32.

For a given flags value, this function always returns the same value.

```
func PdevAuxCount (  
    flags : RmiPdevFlags) => integer
```

B3.48 PdevFlags function

Get RmiPdevFlags value.

```
func PdevFlags (  
    pdev : RmmPdev) => RmiPdevFlags  
begin  
    var flags : RmiPdevFlags;  
  
    case pdev.prot of  
        when PDEV_IOCOH_E2E_IDE => flags.prot = RMI_PDEV_IOCOH_E2E_IDE;  
        when PDEV_IOCOH_E2E_SYS => flags.prot = RMI_PDEV_IOCOH_E2E_SYS;  
        when PDEV_FCOH_E2E_IDE  => flags.prot = RMI_PDEV_FCOH_E2E_IDE;  
        when PDEV_FCOH_E2E_SYS  => flags.prot = RMI_PDEV_FCOH_E2E_SYS;  
    end  
  
    return flags;  
end
```

B3.49 PdevVsmmuCompatible function

Returns TRUE if the attributes of `vsmmu` are compatible with `pdev`.

If the PSMMU associated with the PDEV does not support two stages of translation, this function returns FALSE.

```
func PdevVsmmuIsCompatible(  
    pdev : RmmPdev,
```

```
vsmmu : RmmVsmmu) => boolean
```

B3.50 PlaneRegsValid function

Whether encoding identifies a Plane register.

```
func PlaneRegIsValid(  
    realm : RmmRealm,  
    encoding : bits(64)) => boolean
```

B3.51 PlaneRegValue function

Value of a Plane register.

```
func PlaneRegValue(  
    realm : RmmRealm,  
    plane_idx : integer,  
    encoding : bits(64)) => bits(64)
```

B3.52 PsciReturnCodeEncode function

Return encoding for a PsciReturnCode value.

```
func PsciReturnCodeEncode(  
    value : PsciReturnCode) => bits(64)
```

B3.53 PsciReturnCodePermitted function

Whether a PSCI return code is permitted.

```
func PsciReturnCodePermitted(  
    calling_rec : RmmRec,  
    target_rec : RmmRec,  
    value : PsciReturnCode) => boolean  
begin  
    if value == PSCI_SUCCESS then  
        return TRUE;  
    end  
  
    var fid : bits(64) = calling_rec.gprs[0];  
  
    // Host is permitted to deny a PSCI_CPU_ON request, if the target  
    // CPU is not already on.  
    if (fid == FID_PSCI_CPU_ON  
        && target_rec.flags.runnable != RUNNABLE  
        && value == PSCI_DENIED) then  
        return TRUE;  
    end  
  
    return FALSE;  
end
```

See also:

- [A4.3.7 REC exit due to PSCI](#)
- [B4.3.21 RMI_PSCI_COMPLETE command](#)

B3.54 PsmmuAddrIsValid function

Returns TRUE if `addr` is the base address of a PSMMU.

```
func PsmmuAddrIsValid(  
    addr : Address) => boolean
```

B3.55 PsmmuSupportsMsi function

Returns TRUE if the PSMMU located at `addr` supports MSI.

```
func PsmmuSupportsMsi(  
    addr : Address) => boolean
```

B3.56 RdevFromInstId function

Returns the RDEV identified by `inst_id` and assigned to `realm`.

```
func RdevFromInstId(  
    realm : RmmRealm,  
    inst_id : integer) => RmmRdev
```

B3.57 RdevFromVdevId function

Returns any RDEV identified by `vdev_id` and assigned to `realm`.

```
func RdevFromVdevId(  
    realm : RmmRealm,  
    vdev_id : bits(64)) => RmmRdev
```

B3.58 RdevIdsValid function

Returns TRUE if `vdev_id` identifies a Realm device which is assigned to `realm`.

```
func RdevIdsValid(  
    realm : RmmRealm,  
    vdev_id : bits(64)) => boolean
```

B3.59 RdevIdsAreValid function

Returns TRUE if the tuple (`vdev_id`, `inst_id`) identifies a Realm device which is assigned to `realm`.

```
func RdevIdsAreValid(  
    realm : RmmRealm,  
    vdev_id : bits(64),  
    inst_id : integer) => boolean
```

B3.60 ReadMemory function

Read contents of memory at address range [`addr + offset`, `addr + offset + size`)

`offset` and `size` are both numbers of bytes.

```
func ReadMemory(  
    addr : bits(64),  
    offset : integer,  
    size : integer) => bits(size * 8)
```

B3.61 RealmAt function

Returns the Realm whose RD is located at physical address `addr`.

```
func RealmAt(  
    addr : Address) => RmmRealm
```

See also:

- [A2.1 Realm](#)

B3.62 RealmIsLive function

Returns TRUE if the Realm whose RD is located at physical address `addr` is live.

```
func RealmIsLive(  
    addr : Address) => boolean
```

See also:

- [A2.1.4 Realm liveness](#)

B3.63 RealmParamsSupported function

Returns TRUE if the Realm parameters are supported by the implementation.

```
func RealmParamsSupported(  
    params : RmiRealmParams) => boolean  
begin  
    var impl : RmmFeatures = ImplFeatures();  
  
    if (params.flags0.lpa2 == RMI_FEATURE_TRUE  
        && impl.feats_lpa2 != FEATURE_TRUE) then  
        return FALSE;  
    end  
  
    if (params.flags0.sve == RMI_FEATURE_TRUE  
        && impl.feats_sve != FEATURE_TRUE) then  
        return FALSE;  
    end  
  
    if (params.flags0.pmu == RMI_FEATURE_TRUE  
        && impl.feats_pmu != FEATURE_TRUE) then  
        return FALSE;  
    end  
  
    if (params.flags0.da == RMI_FEATURE_TRUE  
        && impl.feats_da != FEATURE_TRUE) then  
        return FALSE;  
    end  
  
    if (params.s2sz > impl.max_ipa_width) then  
        return FALSE;  
    end  
  
    if (params.sve_vl > impl.max_sve_vl) then  
        return FALSE;  
    end
```

```

    if (params.num_bps == 0
        || params.num_bps + 1 > impl.num_bps) then
        return FALSE;
    end

    if (params.num_wps == 0
        || params.num_wps + 1 > impl.num_wps) then
        return FALSE;
    end

    if (params.pmu_num_ctrs > impl.pmu_num_ctrs) then
        return FALSE;
    end

    if (params.hash_algo == RMI_HASH_SHA_256
        && impl.feats_sha_256 != FEATURE_TRUE) then
        return FALSE;
    end

    if (params.hash_algo == RMI_HASH_SHA_512
        && impl.feats_sha_512 != FEATURE_TRUE) then
        return FALSE;
    end

    if (params.num_aux_planes > impl.max_num_aux_planes) then
        return FALSE;
    end

    if (params.num_aux_planes > 0) then
        if (params.flags1.rtt_tree_per_plane == RMI_FEATURE_FALSE
            && impl.rtt_plane == RTT_PLANE_AUX) then
            return FALSE;
        end

        if (params.flags1.rtt_tree_per_plane == RMI_FEATURE_TRUE
            && impl.rtt_plane == RTT_PLANE_SINGLE) then
            return FALSE;
        end

        if (params.flags1.rtt_s2ap_encoding == RMI_S2AP_INDIRECT
            && impl.rtt_s2ap_indirect == FEATURE_FALSE) then
            return FALSE;
        end

        if ((params.flags1.rtt_tree_per_plane == RMI_FEATURE_TRUE
            && params.flags1.rtt_s2ap_encoding == RMI_S2AP_DIRECT)
            || (params.flags1.rtt_tree_per_plane == RMI_FEATURE_FALSE
            && params.flags1.rtt_s2ap_encoding == RMI_S2AP_INDIRECT)
        ) then
            return FALSE;
        end
    end

    return TRUE;
end

```

See also:

- [A2.1.6 Realm parameters](#)
- [Chapter A3 Feature discovery and configuration](#)

B3.64 RealmRttBaseEqual function

Returns TRUE if RTT base values of `realm` match the provided values.

```
func RealmRttBaseEqual(  
    realm : RmmRealm,  
    rtt_base : Address,  
    aux_rtt_base : array[3] of Address) => boolean  
begin  
    if (realm.rtt_base[0] != rtt_base) then  
        return FALSE;  
    end  
  
    for i = 0 to 2 do  
        if (realm.rtt_base[i + 1] != aux_rtt_base[i]) then  
            return FALSE;  
        end  
    end  
  
    return TRUE;  
end
```

B3.65 RealmVmidEqual function

Returns TRUE if RTT base values of `realm` match the provided values.

```
func RealmVmidEqual(  
    realm : RmmRealm,  
    vmid : bits(16),  
    aux_vmid : array[3] of bits(16)) => boolean  
begin  
    if (realm.vmid[0] != vmid) then  
        return FALSE;  
    end  
  
    for i = 0 to 2 do  
        if (realm.vmid[i + 1] != aux_vmid[i]) then  
            return FALSE;  
        end  
    end  
  
    return TRUE;  
end
```

B3.66 RecAt function

Returns the REC object located at physical address `addr`.

```
func RecAt(  
    addr : Address) => RmmRec
```

See also:

- [A2.3 Realm Execution Context](#)

B3.67 RecAuxCount function

Returns the number of auxiliary Granules required for a REC in the Realm described by `rd`.

The return value is IMPLEMENTATION DEFINED based on the attributes of the Realm.

The return value is guaranteed not to be greater than 16.

For a given Realm, this function always returns the same value.

```
func RecAuxCount (
    rd : Address) => integer
```

B3.68 RecDevMemResponseToRsi function

Returns response to Device memory mapping validation request.

```
func RecDevMemResponseToRsi (
    rec : RmmRec) => RsiResponse
begin
    if ((rec.dev_mem_addr != rec.dev_mem_top)
        && (rec.dev_mem_response == REJECT)) then
        return RSI_REJECT;
    end

    return RSI_ACCEPT;
end
```

B3.69 RecFromMpidr function

Returns the REC object identified by the specified MPIDR value, in the current Realm.

```
func RecFromMpidr (
    mpidr : bits(64)) => RmmRec
```

B3.70 RecIndex function

Returns the REC index which corresponds to mpidr.

```
func RecIndex (
    mpidr : RmiRecMpidr) => integer
begin
    return (UInt(mpidr.aff0)
        + 16 * UInt(mpidr.aff1)
        + 16 * 256 * UInt(mpidr.aff2)
        + 16 * 256 * 256 * UInt(mpidr.aff3));
end
```

See also:

- [A2.3.3 REC index and MPIDR value](#)

B3.71 RecRipasResponseToRsi function

Returns response to RIPAS change request.

```
func RecRipasResponseToRsi (
    rec : RmmRec) => RsiResponse
begin
    if ((rec.ripas_value == RAM)
        && (rec.ripas_addr != rec.ripas_top)
        && (rec.ripas_response == REJECT)) then
        return RSI_REJECT;
    end
```

```
    return RSI_ACCEPT;  
end
```

See also:

- [A5.4 RIPAS change](#)

B3.72 RecS2APResponseToRsi function

Returns response to S2AP change request.

```
func RecS2APResponseToRsi(  
    rec : RmmRec) => RsiResponse  
begin  
    if ((rec.s2ap_addr != rec.s2ap_top)  
        && (rec.s2ap_response == REJECT)) then  
        return RSI_REJECT;  
    end  
  
    return RSI_ACCEPT;  
end
```

See also:

- [A10.3.3.2 Stage 2 Access Permissions change within a multi-Plane Realm](#)

B3.73 RemExtend function

Extend a REM.

The input to the hash function is constructed by concatenating the following, to form a 1024-bit value:

- old_value
- The least significant size bits from new_value
- 512 - size zero bits

This value is hashed using the algorithm identified by hash_algo.

```
func RemExtend(  
    hash_algo : RmmHashAlgorithm,  
    old_value : RmmRealmMeasurement,  
    new_value : RmmRealmMeasurement,  
    size : integer) => RmmRealmMeasurement
```

See also:

- [A7.1.2 Realm Extensible Measurement](#)

B3.74 ResultEqual function

Returns TRUE if command result matches the stated value.

```
func ResultEqual(  
    result : RmiCommandReturnCode,  
    status : RmiStatusCode) => boolean  
  
func ResultEqual(  
    result : RmiCommandReturnCode,  
    status : RmiStatusCode,
```



```
index : integer) => boolean
```

B3.75 RimExtendData function

Extend RIM with contribution from DATA creation.

```
func RimExtendData(  
    realm : RmmRealm,  
    ipa : Address,  
    data : Address,  
    flags : RmiDataFlags) => RmmRealmMeasurement
```

See also:

- [B4.3.1.4 RMI_DATA_CREATE extension of RIM](#)

B3.76 RimExtendRec function

Extend RIM with contribution from REC creation.

```
func RimExtendRec(  
    realm : RmmRealm,  
    params : RmiRecParams) => RmmRealmMeasurement
```

See also:

- [B4.3.28.4 RMI_REC_CREATE extension of RIM](#)

B3.77 RimExtendRipas function

Extend RIM with contribution from RIPAS change for an IPA range.

```
func RimExtendRipas(  
    realm : RmmRealm,  
    base : Address,  
    top : Address,  
    level : integer) => RmmRealmMeasurement  
begin  
    var rim = realm.measurements[0];  
    var size = RttLevelSize(level);  
    var addr = base;  
  
    while (UInt(addr) < UInt(top)) do  
        rim = RimExtendRipasForEntry(rim, addr, level);  
        addr = ToAddress(UInt(addr) + size);  
    end  
  
    return rim;  
end
```

See also:

- [B4.3.42.4 RMI_RTT_INIT_RIPAS extension of RIM](#)

B3.78 RimExtendRipasForEntry function

Extend RIM with contribution from RIPAS change for a single RTT entry.

```
func RimExtendRipasForEntry(  
    rim : RmmRealmMeasurement,
```

```
rim : RmmRealmMeasurement,  
ipa : Address,  
level : integer) => RmmRealmMeasurement
```

B3.79 RimInit function

Initialize RIM.

```
func RimInit(  
    hash_algo : RmmHashAlgorithm,  
    params : RmiRealmParams) => RmmRealmMeasurement
```

See also:

- [B4.3.25.4 RMI_REALM_CREATE initialization of RIM](#)

B3.80 RipasToRmi function

Encodes a RIPAS value.

```
func RipasToRmi(  
    ripas : RmmRipas) => RmiRipas  
begin  
    case ripas of  
        when EMPTY      => return RMI_EMPTY;  
        when RAM         => return RMI_RAM;  
        when DESTROYED   => return RMI_DESTROYED;  
        when DEV         => return RMI_DEV;  
    end  
end
```

B3.81 RmiAddressRangesEqual16 function

Returns TRUE if the first count entries in two arrays of address ranges are equal.

```
func RmiAddressRangesEqual16(  
    ranges1 : array [16] of RmmAddressRange,  
    ranges2 : array [16] of RmiAddressRange,  
    count : integer) => boolean  
begin  
    assert 0 <= count && count <= 16;  
    for i = 0 to count - 1 do  
        if ranges1[i].base != ranges2[i].base then  
            return FALSE;  
        end  
        if ranges1[i].top != ranges2[i].top then  
            return FALSE;  
        end  
    end  
    return TRUE;  
end
```

B3.82 RmiAddressRangesEqual4 function

Returns TRUE if the first count entries in two arrays of address ranges are equal.

```
func RmiAddressRangesEqual4(  
    ranges1 : array [4] of RmmAddressRange,
```

```
        ranges2 : array [4] of RmiAddressRange,  
        count : integer) => boolean  
begin  
    assert 0 <= count && count <= 4;  
    for i = 0 to count - 1 do  
        if ranges1[i].base != ranges2[i].base then  
            return FALSE;  
        end  
        if ranges1[i].top != ranges2[i].top then  
            return FALSE;  
        end  
    end  
    return TRUE;  
end
```

B3.83 RmiDevCommDataAt function

Returns device communication data structure stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
func RmiDevCommDataAt (  
    addr : Address) => RmiDevCommData
```

B3.84 RmiFeatureRegister0Decode function

Decode RmiFeatureRegister0 value.

```
func RmiFeatureRegister0Decode (  
    value : bits(64)) => RmiFeatureRegister0
```

B3.85 RmiFeatureRegisterEncode function

Encode feature register.

```
func RmiFeatureRegisterEncode (  
    index : integer) => bits(64)  
begin  
    var impl : RmmFeatures = ImplFeatures();  
    var result : bits(64) = Zeros();  
  
    if (index == 0) then  
        var reg : RmiFeatureRegister0;  
  
        reg.S2SZ = impl.max_ipa_width;  
        reg.LPA2 = FeatureToRmi(impl.feats_lpa2);  
        reg.SVE = FeatureToRmi(impl.feats_sve);  
        reg.SVE_VL = impl.max_sve_vl;  
  
        assert impl.num_bps >= 2 && impl.num_bps <= 2^6;  
        reg.NUM_BPS = impl.num_bps - 1;  
  
        assert impl.num_wps >= 2 && impl.num_wps <= 2^6;  
        reg.NUM_WPS = impl.num_wps - 1;  
  
        reg.PMU = FeatureToRmi(impl.feats_pmu);  
        reg.PMU_NUM_CTRS = impl.pmu_num_ctrs;  
        reg.HASH_SHA_256 = FeatureToRmi(impl.feats_sha_256);  
        reg.HASH_SHA_512 = FeatureToRmi(impl.feats_sha_512);
```

```

reg.DA = FeatureToRmi(impl.featt_da);

case impl.rtt_plane of
  when RTT_PLANE_AUX          => reg.RTT_PLANE = RMI_RTT_PLANE_AUX;
  when RTT_PLANE_AUX_SINGLE => reg.RTT_PLANE = RMI_RTT_PLANE_AUX_SINGLE;
  when RTT_PLANE_SINGLE      => reg.RTT_PLANE = RMI_RTT_PLANE_SINGLE;
end

reg.RTT_S2AP_INDIRECT = FeatureToRmi(impl.rtt_s2ap_indirect);

reg.MAX_NUM_AUX_PLANES = impl.max_num_aux_planes;
reg.MAX_RECS_ORDER = impl.max_recs_order;

assert impl.gicv3_num_lrs >= 1 && impl.gicv3_num_lrs <= 2^4;
reg.GICV3_NUM_LRS = impl.gicv3_num_lrs - 1;

// Omitted: encode reg into bits(64) value
end

if (index == 1) then
  var reg : RmiFeatureRegister1;

  reg.MAX_MECID = impl.max_mecid;

  // Omitted: encode reg into bits(64) value
end

return result;
end

```

B3.86 RmiPdevEventIsValid function

Returns TRUE if *ev* is a valid encoding, and the event is supported by *pdev*.

```

func RmiPdevEventIsValid(
  ev : RmiPdevEvent) => boolean
begin
  return (ev == RMI_IDE_KEY_REFRESH);
end

```

B3.87 RmiPdevFlagsDecode function

Decode RmiPdevFlags value.

```

func RmiPdevFlagsDecode(
  value : bits(64)) => RmiPdevFlags

```

B3.88 RmiPdevParamsAt function

Returns PDEV parameters stored at physical address *addr*.

If the PAS of *addr* is not NS, the return value is UNKNOWN.

```

func RmiPdevParamsAt(
  addr : Address) => RmiPdevParams

```

B3.89 RmiPdevParamsIsValid function

Returns TRUE if the memory location contains a valid encoding of the RmiPdevParams type and all the following are true:

- The device identifier is valid
- The device identifier is not equal to the device identifier of another PDEV
- The Root Port identifier is valid
- The IDE stream identifier is valid
- The RID range is valid
- The RID range does not overlap the RID range of another PDEV
- The base and top of every address range is aligned to the size of a Granule
- Every address range falls within a memory range permitted by the system
- None of the address ranges overlaps another address range for this PDEV
- None of the address ranges overlaps an address range for another PDEV

```
func RmiPdevParamsIsValid(  
    addr : Address) => boolean
```

B3.90 RmiPublicKeyParamsAt function

Returns public key parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
func RmiPublicKeyParamsAt (  
    addr : Address) => RmiPublicKeyParams
```

B3.91 RmiRealmParamsAt function

Returns Realm parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
func RmiRealmParamsAt (  
    addr : Address) => RmiRealmParams
```

See also:

- [A2.1.6 Realm parameters](#)

B3.92 RmiRealmParamsIsValid function

Returns TRUE if the memory location contains a valid encoding of the RmiRealmParams type.

```
func RmiRealmParamsIsValid(  
    addr : Address) => boolean
```

B3.93 RmiRecParamsAt function

Returns REC parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
func RmiRecParamsAt (  
    addr : Address) => RmiRecParams
```

B3.94 RmiRecRunAt function

Returns the RecRun object stored at physical address `addr`.

```
func RmiRecRunAt (  
    addr : Address) => RmiRecRun
```

See also:

- [A4.2 REC entry](#)
- [A4.3 REC exit](#)

B3.95 RmiVdevFlagsDecode function

Decode RmiVdevFlags value.

```
func RmiVdevFlagsDecode(  
    value : bits(64)) => RmiVdevFlags
```

B3.96 RmiVdevParamsAt function

Returns VDEV parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
func RmiVdevParamsAt (  
    addr : Address) => RmiVdevParams
```

B3.97 RmiVdevParamsIsValid function

Returns TRUE if the memory location contains a valid encoding of the RmiPdevParams type.

```
func RmiVdevParamsIsValid(  
    addr : Address) => boolean
```

B3.98 RmiVersionHigherIsSupported function

Returns TRUE if the RMM supports an RMI revision which is incompatible with and greater than `version`.

```
func RmiVersionHigherIsSupported(  
    version : RmiInterfaceVersion) => boolean
```

See also:

- [Chapter B2 Interface versioning](#)
- [B4.3.56 RMI_VERSION command](#)

B3.99 RmiVersionHighest function

Returns the highest RMI revision supported by the RMM.

```
func RmiVersionHighest() => RmiInterfaceVersion
```

See also:

- [Chapter B2 Interface versioning](#)
- [B4.3.56 RMI_VERSION command](#)

B3.100 RmiVersionHighestBelow function

Returns the highest RMI revision which is both less than `version` and supported by the RMM.

```
func RmiVersionHighestBelow(  
    version : RmiInterfaceVersion) => RmiInterfaceVersion
```

See also:

- [Chapter B2 Interface versioning](#)
- [B4.3.56 RMI_VERSION command](#)

B3.101 RmiVersionIsSupported function

Returns TRUE if the RMM supports an RMI revision which is compatible with `version`.

```
func RmiVersionIsSupported(  
    version : RmiInterfaceVersion) => boolean
```

See also:

- [Chapter B2 Interface versioning](#)
- [B4.3.56 RMI_VERSION command](#)

B3.102 RmiVersionLowerIsSupported function

Returns TRUE if the RMM supports an RMI revision which is incompatible with and less than `version`.

```
func RmiVersionLowerIsSupported(  
    version : RmiInterfaceVersion) => boolean
```

See also:

- [Chapter B2 Interface versioning](#)
- [B4.3.56 RMI_VERSION command](#)

B3.103 RmiVsmmuParamsAt function

Returns VSMMU parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
func RmiVsmmuParamsAt(  
    addr : Address) => RmiVsmmuParams
```

See also:

- [A9.6.3 VSMMU liveness](#)

B3.104 RmiVsmmuParamsIsValid function

Returns TRUE if the memory location contains a valid encoding of the `RmiVsmmuParams` type and all the following are true:

- `aidr` value is not set to an architecturally valid value.
- Any `idr[]` value is not set to an architecturally valid value.

Here, “architecturally valid” refers to the SMMU implementation to which this VSMMU belongs.

```
func RmiVsmmuParamsIsValid(  

```

```
addr : Address) => boolean
```

See also:

- [Arm System Memory Management Unit Architecture Specification \[16\]](#)
- [A9.6 Virtual SMMU](#)

B3.105 RsiDeviceInfoAt function

Returns device configuration stored at IPA `addr`, mapped in the current Realm.

```
func RsiDeviceInfoAt(  
    addr : Address) => RsiDeviceInfo
```

B3.106 RsiFeatureRegisterEncode function

Encode feature register.

```
func RsiFeatureRegisterEncode(  
    realm : RmmRealm,  
    index : integer) => bits(64)  
begin  
    var result : bits(64) = Zeros();  
  
    if (index == 0) then  
        var reg : RsiFeatureRegister0;  
  
        reg.DA = FeatureToRsi(realm.feat_da);  
  
        // Omitted: set reg.MRO depending on whether platform  
        // implements FEAT_S2PIE  
  
        // Omitted: encode reg into bits(64) value  
    end  
  
    return result;  
end
```

B3.107 RsiHostCallAt function

Returns Host call data stored at IPA `addr`, mapped in the current Realm.

```
func RsiHostCallAt(  
    addr : Address) => RsiHostCall
```

B3.108 RsiPlaneRunAt function

Returns the PlaneRun object stored at IPA `addr`.

```
func RsiPlaneRunAt(  
    realm : RmmRealm,  
    addr : Address) => RsiPlaneRun
```

B3.109 RsiRealmConfigAt function

Returns Realm configuration stored at IPA `addr`, mapped in the current Realm.

```
func RsiRealmConfigAt(  
    addr : Address) => RsiRealmConfig
```

```
addr : Address) => RsiRealmConfig
```

B3.110 RsiVersionHigherIsSupported function

Returns TRUE if the RMM supports an RSI revision which is incompatible with and greater than *version*.

```
func RsiVersionHigherIsSupported(  
    version : RsiInterfaceVersion) => boolean
```

See also:

- [Chapter B2 Interface versioning](#)
- [B5.3.26 RSI_VERSION command](#)

B3.111 RsiVersionHighest function

Returns the highest RSI revision supported by the RMM.

```
func RsiVersionHighest() => RsiInterfaceVersion
```

See also:

- [Chapter B2 Interface versioning](#)
- [B5.3.26 RSI_VERSION command](#)

B3.112 RsiVersionHighestBelow function

Returns the highest RSI revision which is both less than *version* and supported by the RMM.

```
func RsiVersionHighestBelow(  
    version : RsiInterfaceVersion) => RsiInterfaceVersion
```

See also:

- [Chapter B2 Interface versioning](#)
- [B5.3.26 RSI_VERSION command](#)

B3.113 RsiVersionIsSupported function

Returns TRUE if the RMM supports an RSI revision which is compatible with *version*.

```
func RsiVersionIsSupported(  
    version : RsiInterfaceVersion) => boolean
```

See also:

- [Chapter B2 Interface versioning](#)
- [B5.3.26 RSI_VERSION command](#)

B3.114 RsiVersionLowerIsSupported function

Returns TRUE if the RMM supports an RSI revision which is incompatible with and less than *version*.

```
func RsiVersionLowerIsSupported(  
    version : RsiInterfaceVersion) => boolean
```

See also:

- [Chapter B2 Interface versioning](#)

- [B5.3.26 RSI_VERSION command](#)

B3.115 RttAllEntriesContiguous function

Returns TRUE if all entries in the RTT at address `rtt` at level `level` have contiguous output addresses, starting with `addr`.

```
func RttAllEntriesContiguous(  
    rtt : RmmRtt,  
    addr : Address,  
    level : integer) => boolean
```

See also:

- [A5.6 Realm Translation Table](#)

B3.116 RttAllEntriesRipas function

Returns TRUE if all entries in the RTT at address `rtt` have RIPAS `ripas`.

```
func RttAllEntriesRipas(  
    rtt : RmmRtt,  
    ripas : RmmRipas) => boolean
```

B3.117 RttAllEntriesState function

Returns TRUE if all entries in the RTT at address `rtt` have state `state`.

```
func RttAllEntriesState(  
    rtt : RmmRtt,  
    state : RmmRttEntryState) => boolean
```

See also:

- [A5.6 Realm Translation Table](#)

B3.118 RttAt function

Returns the RTT at address `rtt`.

```
func RttAt(  
    addr : Address) => RmmRtt
```

B3.119 RttConfigIsValid function

Returns TRUE if the RTT configuration values provided are self-consistent and are supported by the platform.

```
func RttConfigIsValid(  
    ipa_width : integer,  
    rtt_level_start : integer,  
    rtt_num_start : integer) => boolean
```

See also:

- [A5.6 Realm Translation Table](#)

B3.120 RttDescriptorDecode function

Decode an RTT descriptor.

```
func RttDescriptorDecode(  
    desc : bits(64),  
    encoding : RmmRttS2APEncoding) => RmmRttEntry
```

B3.121 RttDescriptorIsValidForUnprotected function

Returns TRUE if, within the descriptor `desc`, all of the following are true:

- All fields which are *Host-controlled Unprotected RTT attributes* are set to architecturally valid values.
- All fields which are not *Host-controlled Unprotected RTT attributes* are set to zero.

```
func RttDescriptorIsValidForUnprotected(  
    desc : bits(64)) => boolean
```

See also:

- [A5.6.12.3 Memory attributes for ASSIGNED_NS mappings](#)

B3.122 RttEntriesInRangeCohDevMem function

Returns TRUE if all entries in the RTT at address `rtt` at level `level`, within Protected IPA range [`base`, `top`), have output addresses which map to coherent device memory.

```
func RttEntriesInRangeCohDevMem(  
    rtt : RmmRtt,  
    level : integer,  
    base : Address,  
    top : Address) => boolean  
begin  
    var addr : Address = base;  
    var size : integer = RttLevelSize(level);  
  
    while (UInt(addr) < UInt(top)) do  
        var index : integer = RttEntryIndex(addr, level);  
        var rtte : RmmRttEntry = RttEntryAt(rtt, index);  
  
        if (!PaIsDelegableCohDevMem(rtte.addr)) then  
            return FALSE;  
        end  
  
        addr = ToAddress(UInt(addr) + size);  
    end  
  
    return TRUE;  
end
```

B3.123 RttEntriesInRangeMemAttr function

Returns TRUE if all entries in the RTT at address `rtt` at level `level`, within Protected IPA range [`base`, `top`), have memory attributes `attr`.

```
func RttEntriesInRangeMemAttr(  
    rtt : RmmRtt,  
    level : integer,  
    base : Address,
```

```

    top : Address,
    attr : RmmRttMemAttr) => boolean
begin
    var addr : Address = base;
    var size : integer = RttLevelSize(level);

    while (UInt(addr) < UInt(top)) do
        var index : integer = RttEntryIndex(addr, level);
        var rtte : RmmRttEntry = RttEntryAt(rtt, index);

        if (rtte.attr_prot != attr) then
            return FALSE;
        end

        addr = ToAddress(UInt(addr) + size);
    end

    return TRUE;
end

```

B3.124 RttEntriesInRangeNonCohDevMem function

Returns TRUE if all entries in the RTT at address `rtt` at level `level`, within Protected IPA range [base, top), have output addresses which map to non-coherent device memory.

```

func RttEntriesInRangeNonCohDevMem(
    rtt : RmmRtt,
    level : integer,
    base : Address,
    top : Address) => boolean
begin
    var addr : Address = base;
    var size : integer = RttLevelSize(level);

    while (UInt(addr) < UInt(top)) do
        var index : integer = RttEntryIndex(addr, level);
        var rtte : RmmRttEntry = RttEntryAt(rtt, index);

        if (!PaIsDelegableNonCohDevMem(rtte.addr)) then
            return FALSE;
        end

        addr = ToAddress(UInt(addr) + size);
    end

    return TRUE;
end

```

B3.125 RttEntriesInRangeOutputContiguous function

Returns TRUE if all entries in the RTT at address `rtt` at level `level`, within IPA range [base, top), map to a contiguous range of output addresses starting from `out`.

```

func RttEntriesInRangeOutputContiguous(
    rtt : RmmRtt,
    level : integer,
    base : Address,
    top : Address,
    out : Address) => boolean

```

```
begin
    var in_addr : Address = base;
    var out_addr : Address = out;
    var size : integer = RttLevelSize(level);

    while (UInt(in_addr) < UInt(top)) do
        var index : integer = RttEntryIndex(in_addr, level);
        var rtte : RmmRttEntry = RttEntryAt(rtt, index);

        if (rtte.addr != out_addr) then
            return FALSE;
        end

        in_addr = ToAddress(UInt(in_addr) + size);
        out_addr = ToAddress(UInt(out_addr) + size);
    end

    return TRUE;
end
```

B3.126 RttEntriesInRangeRipas function

Returns TRUE if all entries in the RTT at address `rtt` at level `level`, within IPA range [base, top), have RIPAS `ripas`.

```
func RttEntriesInRangeRipas(
    rtt : RmmRtt,
    level : integer,
    base : Address,
    top : Address,
    ripas : RmmRipas) => boolean
begin
    var addr : Address = base;
    var size : integer = RttLevelSize(level);

    while (UInt(addr) < UInt(top)) do
        var index : integer = RttEntryIndex(addr, level);
        var rtte : RmmRttEntry = RttEntryAt(rtt, index);

        if (rtte.ripas != ripas) then
            return FALSE;
        end

        addr = ToAddress(UInt(addr) + size);
    end

    return TRUE;
end
```

B3.127 RttEntryAt function

Returns the `i`th entry in the RTT `rtt`.

```
func RttEntryAt(
    rtt : RmmRtt,
    i : integer) => RmmRttEntry
```

See also:

- [A5.6 Realm Translation Table](#)

B3.128 RttEntryIndex function

Returns the index of the entry in a level `level` RTT which is identified by `addr`.

```
func RttEntryIndex(  
    addr : Address,  
    level : integer) => integer
```

See also:

- [A5.6 Realm Translation Table](#)

B3.129 RttEntryStateToRmi function

Encodes the state of an RTTE.

```
func RttEntryStateToRmi(  
    state : RmmRttEntryState) => RmiRttEntryState  
begin  
    case state of  
        when UNASSIGNED           => return RMI_UNASSIGNED;  
        when ASSIGNED             => return RMI_ASSIGNED;  
        when UNASSIGNED_NS        => return RMI_UNASSIGNED;  
        when ASSIGNED_NS          => return RMI_ASSIGNED;  
        when TABLE               => return RMI_TABLE;  
        when ASSIGNED_DEV         => return RMI_ASSIGNED_DEV;  
        when AUX_DESTROYED        => return RMI_AUX_DESTROYED;  
        when ASSIGNED_VSMMU       => return RMI_ASSIGNED_VSMMU;  
    end  
end
```

B3.130 RttFold function

Returns the RTTE which results from folding the homogeneous RTT at address `rtt`.

```
func RttFold(  
    rtt : RmmRtt) => RmmRttEntry
```

See also:

- [A5.6.6 RTT folding](#)

B3.131 RttIsHomogeneous function

Returns TRUE if the RTT at address `rtt` is homogeneous.

```
func RttIsHomogeneous(  
    rtt : RmmRtt) => boolean
```

See also:

- [A5.6.6 RTT folding](#)

B3.132 RttIsLive function

Returns TRUE if the RTT at address `rtt` is live.

```
func RttIsLive(  
    rtt : RmmRtt) => boolean
```

See also:

- [A5.6.8 RTTE liveness and RTT liveness](#)
- [A5.6.9 RTT destruction](#)

B3.133 RttLevelsBlockOrPage function

Returns TRUE if *level* is either a block or page RTT level for the Realm described by *rd*.

```
func RttLevelsBlockOrPage(  
    realm : RmmRealm,  
    level : integer) => boolean
```

See also:

- [A5.6 Realm Translation Table](#)

B3.134 RttLevelsStarting function

Returns TRUE if *level* is the starting level of the RTT for the Realm described by *rd*.

```
func RttLevelsStarting(  
    realm : RmmRealm,  
    level : integer) => boolean
```

See also:

- [A5.6 Realm Translation Table](#)

B3.135 RttLevelsValid function

Returns TRUE if *level* is a valid RTT level for the Realm described by *rd*.

```
func RttLevelsValid(  
    realm : RmmRealm,  
    level : integer) => boolean
```

See also:

- [A5.6 Realm Translation Table](#)

B3.136 RttLevelSize function

Returns the size of the address space described by each entry in an RTT at *level*.

If *level* is invalid, the return value is UNKNOWN.

```
func RttLevelSize(  
    level : integer) => integer
```

See also:

- [A5.6 Realm Translation Table](#)

B3.137 RttMemAttrEqual function

Returns TRUE if the memory attributes of the two RTT entries match.

```
func RttMemAttrEqual(  
    rtte1 : RmmRttEntry,  
    rtte2 : RmmRttEntry,  
    prot  : RmmRttProtected) => boolean  
begin  
    case prot of  
        when RTT_PROTECTED =>  
            return rtte1.attr_prot == rtte2.attr_prot;  
        when RTT_UNPROTECTED =>  
            return rtte1.attr_unprot == rtte2.attr_unprot;  
    end  
end
```

B3.138 RttS2APEqual function

Returns TRUE if the S2AP of the two RTT entries match.

```
func RttS2APEqual(  
    rtte1 : RmmRttEntry,  
    rtte2 : RmmRttEntry,  
    encoding : RmmRttS2APEncoding) => boolean  
begin  
    case encoding of  
        when S2AP_DIRECT => return (  
            (rtte1.s2ap_direct.read  
             == rtte2.s2ap_direct.read)  
            && (rtte1.s2ap_direct.write  
              == rtte2.s2ap_direct.write));  
        when S2AP_INDIRECT => return (  
            (rtte1.s2ap_indirect.base_index  
             == rtte2.s2ap_indirect.base_index)  
            && (rtte1.s2ap_indirect.overlay_index  
              == rtte2.s2ap_indirect.overlay_index));  
    end  
end
```

B3.139 RttsAllProtectedEntriesRipas function

Returns TRUE if the RIPAS of all entries identified by Protected IPAs in all of the starting-level RTT Granules is equal to ripas.

```
func RttsAllProtectedEntriesRipas(  
    rtt_base : Address,  
    rtt_num_start : integer,  
    ripas : RmmRipas) => boolean
```

B3.140 RttsAllProtectedEntriesState function

Returns TRUE if the state of all entries identified by Protected IPAs in all of the starting-level RTT Granules is equal to state.

```
func RttsAllProtectedEntriesState(  
    rtt_base : Address,  
    rtt_num_start : integer,
```

```
state : RmmRttEntryState) => boolean
```

B3.141 RttsAllUnprotectedEntriesState function

Returns TRUE if the state of all entries identified by Unprotected IPAs in all of the starting-level RTT Granules is equal to *state*.

```
func RttsAllUnprotectedEntriesState(
    rtt_base : Address,
    rtt_num_start : integer,
    state : RmmRttEntryState) => boolean
```

B3.142 RttsGranuleState function

Inductive function which identifies the states of the starting-level RTT Granules.

This function is used in the definition of command footprint.

```
func RttsGranuleState(
    rtt_base : Address,
    rtt_num_start : integer)
```

B3.143 RttSkipEntriesIfNotState function

Scanning *rtt* starting from *base* and terminating at *top*, returns the IPA of the first entry whose state is not *state*.

If no entry is found whose state is *state*, returns the next IPA after the last entry in *rtt*.

The return value is aligned to the size of the address range described by an entry at RTT *level*.

```
func RttSkipEntriesIfNotState(
    rtt : RmmRtt,
    level : integer,
    base : Address,
    top : Address,
    state : RmmRttEntryState) => Address
```

B3.144 RttSkipEntriesUnlessRipas function

Scanning *rtt* starting from *ipa*, returns the IPA of the first entry whose RIPAS is *ripas*.

If no entry is found whose RIPAS is *ripas*, returns the next IPA after the last entry in *rtt*.

The return value is aligned to the size of the address range described by an entry at RTT *level*.

```
func RttSkipEntriesUnlessRipas(
    rtt : RmmRtt,
    level : integer,
    ipa : Address,
    ripas : RmmRipas) => Address
```

B3.145 RttSkipEntriesUnlessState function

Scanning *rtt* starting from *ipa*, returns the IPA of the first entry whose state is *state*.

If no entry is found whose state is *state*, returns the next IPA after the last entry in *rtt*.

The return value is aligned to the size of the address range described by an entry at RTT *level*.

```
func RttSkipEntriesUnlessState(  
    rtt : RmmRtt,  
    level : integer,  
    ipa : Address,  
    state : RmmRttEntryState) => Address
```

B3.146 RttSkipEntriesWithRipas function

Scan rtt starting from base and terminating at top.

- If stop_at_destroyed is FALSE then return IPA of the first entry whose state is TABLE.
- If stop_at_destroyed is TRUE then return IPA of the first entry whose state is TABLE or whose RIPAS is DESTROYED.

If no such entry is found, returns the smaller of:

- The next IPA after the last entry in rtt
- The top argument.

The return value is aligned to the size of the address range described by an entry at RTT level.

```
func RttSkipEntriesWithRipas(  
    rtt : RmmRtt,  
    level : integer,  
    base : Address,  
    top : Address,  
    stop_at_destroyed : boolean) => Address  
begin  
    var result : Address = RttSkipEntriesUnlessState(  
        rtt, level, base, TABLE);  
  
    if stop_at_destroyed then  
        result = MinAddress(result,  
            RttSkipEntriesUnlessRipas(  
                rtt, level, base, DESTROYED));  
    end  
  
    result = MinAddress(result, top);  
  
    return AlignDownToRttLevel(result, level);  
end
```

B3.147 RttSkipNonLiveEntries function

Scanning rtt starting from ipa, returns the IPA of the first live entry.

If no live entry is found, returns the next IPA after the last entry in rtt.

The return value is aligned to the size of the address range described by an entry at RTT level.

```
func RttSkipNonLiveEntries(  
    rtt : RmmRtt,  
    level : integer,  
    ipa : Address) => Address  
begin  
    var result : Address = RttSkipEntriesUnlessState(  
        rtt, level, ipa, ASSIGNED);  
  
    result = MinAddress(result,  
        RttSkipEntriesUnlessState(  

```

```
        rtt, level, ipa, ASSIGNED_NS));

result = MinAddress(result,
    RttSkipEntriesUnlessState(
        rtt, level, ipa, TABLE));

result = MinAddress(result,
    RttSkipEntriesUnlessState(
        rtt, level, ipa, ASSIGNED_DEV));

return AlignDownToRttLevel(result, level);
end
```

See also:

- [A5.6.8 RTTE liveness and RTT liveness](#)

B3.148 RttsStateEqual function

Returns TRUE if the state of all of the starting-level RTT Granules is equal to *state*.

```
func RttsStateEqual(
    rtt_base : Address,
    rtt_num_start : integer,
    state : RmmGranuleState) => boolean
begin
    for i = 0 to rtt_num_start - 1 do
        var addr = (UInt(rtt_base) + i * RMM_GRANULE_SIZE) [(ADDRESS_WIDTH-1):0];
        if (!PaIsDelegable(addr)
            || GranuleAt(addr).state != state) then
            return FALSE;
        end
    end
    return TRUE;
end
```

B3.149 RttWalk function

Returns the result of an RTT walk from the base of RTT tree *index* owned by *rd*, to address *addr*.

The walk does not progress beyond *level*.

```
func RttWalk(
    realm : RmmRealm,
    addr : Address,
    level : integer,
    index : integer) => RmmRttWalkResult
```

See also:

- [A5.6.10 RTT walk](#)

B3.150 RttWalkAnyNotAligned function

Performs one or more RTT walks within the IPA range [*base*, *top*), on one or more RTT trees owned by *rd*.

For a Realm which is configured to use an RTT tree per Plane, it is permitted for an implementation to either walk just one of the RTTs, or to walk all of them.

It is permitted for an implementation to perform multiple walks, at successive IPAs, on the same RTT.

If one of the walks performed terminates earlier than `level` then the return value indicates the RTT index and the IPA at which the walk was performed. In this case, the result's "valid" value is TRUE.

If none of the walks performed terminates earlier than `level` then the result's "valid" value is FALSE.

```
func RttWalkAnyNotAligned(  
    realm : RmmRealm,  
    base : Address,  
    top : Address,  
    level : integer) => RmmRttWalkNotAligned
```

B3.151 TdildIsFree function

Returns TRUE if `tdi_id` is unused within the segment identified by `segment_id`.

```
func TdiIdIsFree(  
    tdi_id : bits(64),  
    segment_id : bits(8)) => boolean
```

B3.152 ToAddress function

Convert integer to Address.

```
func ToAddress(value : integer) => Address  
begin  
    return value[(ADDRESS_WIDTH-1):0];  
end
```

B3.153 ToBits64 function

Convert integer to Bits64.

```
func ToBits64(value : integer) => bits(64)  
begin  
    return value[63:0];  
end
```

B3.154 VdevAt function

Returns the VDEV object located at physical address `addr`.

```
func VdevAt(  
    addr : Address) => RmmVdev
```

B3.155 VdevAuxCount function

Returns the number of auxiliary Granules required for a VDEV with the specified flags.

The return value is guaranteed not to be greater than 32.

For a given flags value, this function always returns the same value.

```
func VdevAuxCount(  
    pdev_flags : RmiPdevFlags,  
    vdev_flags : RmiVdevFlags) => integer
```

B3.156 VersionEqual function

Returns TRUE if command result matches the stated value.

```
func VersionEqual(  
    ver1 : RmiInterfaceVersion,  
    ver2 : RmiInterfaceVersion) => boolean
```

```
func VersionEqual(  
    ver1 : RsiInterfaceVersion,  
    ver2 : RsiInterfaceVersion) => boolean
```

See also:

- [Chapter B2 Interface versioning](#)

B3.157 VmidsAreFree function

Returns TRUE if vmid is unused.

```
func VmidsAreFree(  
    vmid : array[4] of bits(16)) => boolean
```

```
func VmidsAreFree(  
    vmid : bits(16),  
    aux_vmid : array [3] of bits(16)) => boolean
```

B3.158 VmidsAreValid function

Returns TRUE if vmid is valid on the platform.

```
func VmidsAreValid(  
    vmid : bits(16),  
    aux_vmid : array [3] of bits(16)) => boolean
```

If the underlying hardware platform does not implement FEAT_VMID16 then a VMID value with vmid[15:8] != 0 is invalid.

See also:

- [A2.1.3 Realm attributes](#)
- [B4.3.25 RMI_REALM_CREATE command](#)

B3.159 VsidIsFree function

Returns TRUE if vsid is unused within the VSMMU vsmmu.

```
func VsidIsFree(  
    vsmmu : RmmVsmmu,  
    vsid : bits(64)) => boolean
```

B3.160 VsmmuAt function

Returns the VSMMU object located at physical address addr.

```
func VsmmuAt(  
    addr : Address) => RmmVsmmu
```

See also:

- [A9.6 Virtual SMMU](#)

B3.161 VsmmulIsLive function

Returns TRUE if the VSMMU located at physical address `addr` is live.

```
func VsmmulIsLive(  
    addr : Address) => boolean
```

See also:

- [A9.6.3 VSMMU liveness](#)

DRAFT

Chapter B4

Realm Management Interface

This chapter defines the interface used by the Host to manage Realms.

B4.1 RMI version

R_{NCFDX} This specification defines version 1.1 of the Realm Management Interface.

See also:

- [Chapter B2 Interface versioning](#)
- [B4.3.56 RMI_VERSION command](#)

B4.2 RMI command return codes

I_{JQMBN} The return code of an RMI command is a tuple which contains *status* and *index* fields.

I_{YCHQV} The *status* field of an RMI command return code indicates whether the command

- succeeded, or
- failed, and the reason for the failure.

I_{PPNST} If an RMI command succeeds then the status of its return code is RMI_SUCCESS.

I_{MBVPG} The *index* field of an RMI command return code can provide additional information about the reason for a command failure. The meaning of the index field depends on the status, and is described by the following table.

Status	Description	Meaning of index
RMI_SUCCESS	Command completed successfully	None: index is zero.
RMI_ERROR_INPUT	The value of a command input value caused the command to fail	None: index is zero.
RMI_ERROR_REALM	An attribute of a Realm does not match the expected value	Varies between usages. See individual commands for details.
RMI_ERROR_REC	An attribute of a REC does not match the expected value	None: index is zero.
RMI_ERROR_RTT	An RTT walk terminated before reaching the target RTT level, or reached an RTTE with an unexpected value	RTT level at which the walk terminated.
RMI_ERROR_RTT_AUX	RTTE in an auxiliary RTT contained an unexpected value	In some cases, indicates auxiliary RTT level at which the walk terminated. See individual commands for details.

I_{QQQNB} Multiple failure conditions in an RMI command may return the same error code - that is, the same status and index values.

R_{XRDYQ} If an input to an RMI command uses an invalid encoding then the command fails and returns RMI_ERROR_INPUT. Command inputs include registers and in-memory data structures.

Invalid encodings include:

- using a reserved encoding in an enumeration

See also:

- [B4.4.3 RmiCommandReturnCode type](#)

B4.3 RMI commands

The following table summarizes the FIDs of commands in the RMI interface.

FID	Command
0xC4000150	RMI_VERSION
0xC4000151	RMI_GRANULE_DELEGATE
0xC4000152	RMI_GRANULE_UNDELEGATE
0xC4000153	RMI_DATA_CREATE
0xC4000154	RMI_DATA_CREATE_UNKNOWN
0xC4000155	RMI_DATA_DESTROY
0xC4000156	RMI_PDEV_AUX_COUNT
0xC4000157	RMI_REALM_ACTIVATE
0xC4000158	RMI_REALM_CREATE
0xC4000159	RMI_REALM_DESTROY
0xC400015A	RMI_REC_CREATE
0xC400015B	RMI_REC_DESTROY
0xC400015C	RMI_REC_ENTER
0xC400015D	RMI_RTT_CREATE
0xC400015E	RMI_RTT_DESTROY
0xC400015F	RMI_RTT_MAP_UNPROTECTED
0xC4000160	RMI_VDEV_AUX_COUNT
0xC4000161	RMI_RTT_READ_ENTRY
0xC4000162	RMI_RTT_UNMAP_UNPROTECTED
0xC4000163	RMI_RTT_DEV_MEM_VALIDATE
0xC4000164	RMI_PSCI_COMPLETE
0xC4000165	RMI_FEATURES
0xC4000166	RMI_RTT_FOLD
0xC4000167	RMI_REC_AUX_COUNT
0xC4000168	RMI_RTT_INIT_RIPAS
0xC4000169	RMI_RTT_SET_RIPAS
0xC400016A	RMI_VSMMU_CREATE
0xC400016B	RMI_VSMMU_DESTROY
0xC400016C	RMI_VSMMU_MAP
0xC400016D	RMI_VSMMU_UNMAP
0xC400016E	RMI_PSMMU_MSI_CONFIG
0xC400016F	RMI_PSMMU_IRQ_NOTIFY

FID	Command
...	
0xC4000172	RMI_DEV_MEM_MAP
0xC4000173	RMI_DEV_MEM_UNMAP
0xC4000174	RMI_PDEV_ABORT
0xC4000175	RMI_PDEV_COMMUNICATE
0xC4000176	RMI_PDEV_CREATE
0xC4000177	RMI_PDEV_DESTROY
0xC4000178	RMI_PDEV_GET_STATE
0xC4000179	RMI_PDEV_IDE_RESET
0xC400017A	RMI_PDEV_NOTIFY
0xC400017B	RMI_PDEV_SET_PUBKEY
0xC400017C	RMI_PDEV_STOP
0xC400017D	RMI_RTT_AUX_CREATE
0xC400017E	RMI_RTT_AUX_DESTROY
0xC400017F	RMI_RTT_AUX_FOLD
0xC4000180	RMI_RTT_AUX_MAP_PROTECTED
0xC4000181	RMI_RTT_AUX_MAP_UNPROTECTED
...	
0xC4000183	RMI_RTT_AUX_UNMAP_PROTECTED
0xC4000184	RMI_RTT_AUX_UNMAP_UNPROTECTED
0xC4000185	RMI_VDEV_ABORT
0xC4000186	RMI_VDEV_COMMUNICATE
0xC4000187	RMI_VDEV_CREATE
0xC4000188	RMI_VDEV_DESTROY
0xC4000189	RMI_VDEV_GET_STATE
0xC400018A	RMI_VDEV_STOP
0xC400018B	RMI_RTT_SET_S2AP
0xC400018C	RMI_MEC_SET_SHARED
0xC400018D	RMI_MEC_SET_PRIVATE
0xC400018E	RMI_VDEV_COMPLETE

B4.3.1 RMI_DATA_CREATE command

Creates a Data Granule, copying contents from a Non-secure Granule provided by the caller.

See also:

- [Chapter A5 Realm memory management](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [D1.2.3 Initialize memory of New Realm flow](#)

B4.3.1.1 Interface

B4.3.1.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000153
rd	X1	63:0	Address	PA of the RD for the target Realm
data	X2	63:0	Address	PA of the target Data
ipa	X3	63:0	Address	IPA at which the Granule will be mapped in the target Realm
src	X4	63:0	Address	PA of the source Granule
flags	X5	63:0	RmiDataFlags	Flags

B4.3.1.1.2 Context

The RMI_DATA_CREATE command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index

B4.3.1.1.3 Output values

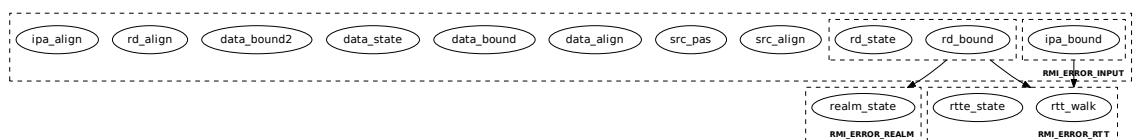
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.1.2 Failure conditions

ID	Condition
src_align	pre: !AddrIsGranuleAligned(src) post: ResultEqual(result, RMI_ERROR_INPUT)
src_pas	pre: !GranuleAccessPermitted(src, PAS_NS) post: ResultEqual(result, RMI_ERROR_INPUT)
data_align	pre: !AddrIsGranuleAligned(data) post: ResultEqual(result, RMI_ERROR_INPUT)
data_bound	pre: !PaIsDelegableDram(data) post: ResultEqual(result, RMI_ERROR_INPUT)
data_state	pre: GranuleAt(data).state != DELEGATED post: ResultEqual(result, RMI_ERROR_INPUT)
data_bound2	pre: ((realm.feat_lpa2 == FEATURE_FALSE) && (UInt(data) >= 2^48)) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsGranuleAligned(ipa) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: !AddrIsProtected(ipa, realm_pre) post: ResultEqual(result, RMI_ERROR_INPUT)
realm_state	pre: realm_pre.state != REALM_NEW post: ResultEqual(result, RMI_ERROR_REALM)
rtt_walk	pre: walk.level < RMM_RTT_PAGE_LEVEL post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_state	pre: walk.rtte.state != UNASSIGNED post: ResultEqual(result, RMI_ERROR_RTT, walk.level)

B4.3.1.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[ipa_bound] < [rtt_walk, rtte_state]
```



B4.3.1.3 Success conditions

ID	Condition
data_state	GranuleAt(data).state == DATA

ID	Condition
data_content	Contents of target Granule are copied from source Granule.
rtte_state	walk.rtte.state == ASSIGNED
rtte_ripas	walk.rtte.ripas == RAM
rtte_addr	walk.rtte.addr == data
rtte_memattr	walk.rtte.attr_prot == MEMATTR_CACHEABLE
rtte_sh	walk.rtte.sh == SHAREABILITY_INNER
rim	realm.measurements[0] == RimExtendData (realm_pre, ipa, data, flags)

B4.3.1.4 RMI_DATA_CREATE extension of RIM

On successful execution of RMI_DATA_CREATE, the new RIM value of the target Realm is calculated by the RMM as follows:

1. If flags.measure == RMI_MEASURE_CONTENT then using the RHA of the target Realm, compute the hash of the contents of the DATA Granule.
2. Allocate an [RmmMeasurementDescriptorData](#) data structure.
3. Populate the measurement descriptor:
 - Set the desc_type field to the descriptor type.
 - Set the len field to the descriptor length.
 - Set the rim field to the current RIM value of the target Realm.
 - Set the ipa field to the IPA at which the DATA Granule is mapped in the target Realm.
 - Set the flags field to the flags provided by the Host.
 - If flags.measure == RMI_MEASURE_CONTENT then set the content field to the hash of the contents of the DATA Granule. Otherwise, set the content field to zero.
4. Using the RHA of the target Realm, compute the hash of the measurement descriptor. Set the RIM of the target Realm to this value, zero filling upper bytes if the RHA output is smaller than the size of the RIM.

See also:

- [A7.1.1 Realm Initial Measurement](#)
- [B3.75 RimExtendData function](#)
- [C2.17 RmmMeasurementDescriptorData type](#)

B4.3.1.5 Footprint

ID	Value
data_state	GranuleAt (data).state
rim	realm.measurements[0]
rtte	RttEntryAt (RttAt (walk.rtt_addr), entry_idx)

B4.3.2 RMI_DATA_CREATE_UNKNOWN command

Creates a Data Granule with unknown contents.

See also:

- [A2.2.5 Granule wiping](#)
- [Chapter A5 Realm memory management](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [D1.5.1 Add memory to Active Realm flow](#)

B4.3.2.1 Interface

B4.3.2.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xc4000154
rd	X1	63:0	Address	PA of the RD for the target Realm
data	X2	63:0	Address	PA of the target Data
ipa	X3	63:0	Address	IPA at which the Granule will be mapped in the target Realm

B4.3.2.1.2 Context

The RMI_DATA_CREATE_UNKNOWN command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index

B4.3.2.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

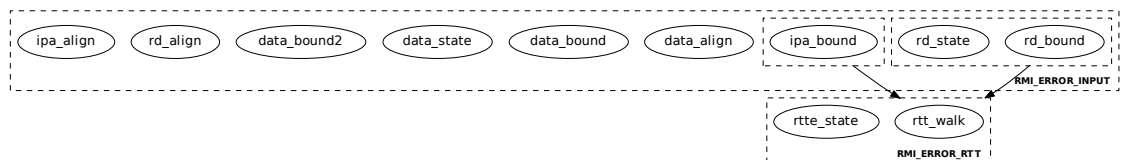
B4.3.2.2 Failure conditions

ID	Condition
data_align	pre: !AddrIsGranuleAligned (data) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
data_bound	pre: !PaIsDelegableDram(data) post: ResultEqual(result, RMI_ERROR_INPUT)
data_state	pre: GranuleAt(data).state != DELEGATED post: ResultEqual(result, RMI_ERROR_INPUT)
data_bound2	pre: ((realm.feat_lpa2 == FEATURE_FALSE) && (UInt(data) >= 2^48)) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsGranuleAligned(ipa) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: !AddrIsProtected(ipa, realm) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < RMM_RTT_PAGE_LEVEL post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_state	pre: walk.rtte.state != UNASSIGNED post: ResultEqual(result, RMI_ERROR_RTT, walk.level)

B4.3.2.2.1 Failure condition ordering

[rd_bound, rd_state] < [rtt_walk, rtte_state]
[ipa_bound] < [rtt_walk, rtte_state]



B4.3.2.3 Success conditions

ID	Condition
data_state	GranuleAt(data).state == DATA
data_content	Contents of target Granule are wiped.
rtte_state	walk.rtte.state == ASSIGNED
rtte_addr	walk.rtte.addr == data
rtte_memattr	walk.rtte.attr_prot == MEMATTR_CACHEABLE
rtte_sh	walk.rtte.sh == SHAREABILITY_INNER

B4.3.2.4 Footprint

ID	Value
data_state	<code>GranuleAt(data).state</code>
rtte	<code>RttEntryAt(RttAt(walk.rtt_addr), entry_idx)</code>

DRAFT

B4.3.3 RMI_DATA_DESTROY command

Destroys a Data Granule.

See also:

- [Chapter A5 Realm memory management](#)
- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [D1.2.5 Realm destruction flow](#)

B4.3.3.1 Interface

B4.3.3.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000155
rd	X1	63:0	Address	PA of the RD which owns the target Data
ipa	X2	63:0	Address	IPA at which the Granule is mapped in the target Realm

B4.3.3.1.2 Context

The RMI_DATA_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
walk_top	Address	RttSkipNonLiveEntries (RttAt (walk.rtt_addr), walk.level, ipa)	false	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.3.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
data	X1	63:0	Address	PA of the Data Granule which was destroyed
top	X2	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

The data output value is valid only when the command result is RMI_SUCCESS.

The values of the `result` and `top` output values for different command outcomes are summarized in the following table.

Scenario	result	top	walk.rtte.state
ipa is mapped as a page and RIPAS is RAM	RMI_SUCCESS	> ipa	Before execution: ASSIGNED and RIPAS is RAM After execution: UNASSIGNED and RIPAS is DESTROYED
ipa is mapped as a page and RIPAS is not RAM	RMI_SUCCESS	> ipa	Before execution: ASSIGNED and RIPAS is not RAM After execution: UNASSIGNED and RIPAS is unchanged
ipa is not mapped	(RMI_ERROR_RTT, <= 3)	> ipa	UNASSIGNED
ipa is mapped as a block	(RMI_ERROR_RTT, 0 < level < 3)	== ipa	ASSIGNED
RTT walk was not performed, due to any other command failure	Another error code	0	Unknown

See also:

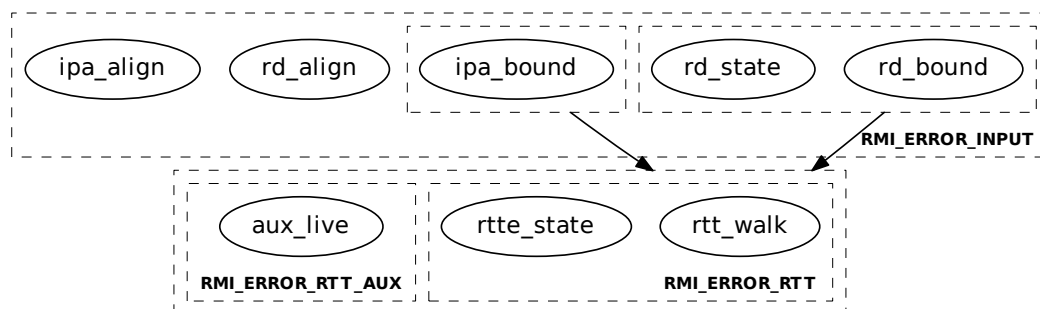
- [A5.6.8 RTTE liveness and RTT liveness](#)

B4.3.3.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsGranuleAligned(ipa) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: !AddrIsProtected(ipa, realm) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < RMM_RTT_PAGE_LEVEL post: (ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))
rtte_state	pre: walk.rtte.state != ASSIGNED post: (ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))
aux_live	pre: AddrIsAuxLive(ipa, realm) post: ResultEqual(result, RMI_ERROR_RTT_AUX, 0)

B4.3.3.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state, aux_live]
[ipa_bound] < [rtt_walk, rtte_state, aux_live]
```



B4.3.3.3 Success conditions

ID	Condition
data_state	<code>GranuleAt(walk.rtte.addr).state == DELEGATED</code>
rtte_state	<code>walk.rtte.state == UNASSIGNED</code>
ripas_ram	pre: <code>walk.rtte.ripas == RAM</code> post: <code>walk.rtte.ripas == DESTROYED</code>
data	<code>data == walk.rtte.addr</code>
top	<code>top == walk_top</code>

B4.3.3.4 Footprint

ID	Value
data_state	<code>GranuleAt(walk.rtte.addr).state</code>
rtte	<code>RttEntryAt(RttAt(walk.rtt_addr), entry_idx)</code>

B4.3.4 RMI_DEV_MEM_MAP command

Maps device memory.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.4.1 Interface

B4.3.4.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000172
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	IPA at which the Granule will be mapped in the target Realm
level	X3	63:0	Int64	RTT level
addr	X4	63:0	Address	PA of the target device memory

B4.3.4.1.2 Context

The RMI_DEV_MEM_MAP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level, RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
gran_state_pre	RmmGranuleState	GranuleAt (addr).state	true	Previous Granule state

B4.3.4.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

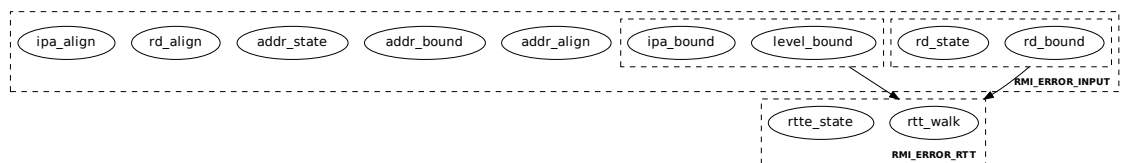
B4.3.4.2 Failure conditions

ID	Condition
addr_align	pre: !AddrIsGranuleAligned (addr) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
addr_bound	pre: !PaIsDelegableDevMem(addr) post: ResultEqual(result, RMI_ERROR_INPUT)
addr_state	pre: GranuleAt(addr).state != DELEGATED post: ResultEqual(result, RMI_ERROR_INPUT)
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
level_bound	pre: !RttLevelIsBlockOrPage(realm, level) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsGranuleAligned(ipa) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: !AddrIsProtected(ipa, realm) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < level post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_state	pre: walk.rtte.state != UNASSIGNED post: ResultEqual(result, RMI_ERROR_RTT, walk.level)

B4.3.4.2.1 Failure condition ordering

[rd_bound, rd_state] < [rtt_walk, rtte_state]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]



B4.3.4.3 Success conditions

ID	Condition
state	GranuleAt(addr).state == DEV_MAPPED
rtte_state	walk.rtte.state == ASSIGNED_DEV
rtte_addr	walk.rtte.addr == addr
rtte_attr_ncoh	pre: PaIsDelegableNonCohDevMem(addr) post: walk.rtte.attr_prot == MEMATTR_NON_CACHEABLE
rtte_attr_coh	pre: PaIsDelegableCohDevMem(addr) post: walk.rtte.attr_prot == MEMATTR_PASSTHROUGH
rtte_sh_ncoh	pre: PaIsDelegableNonCohDevMem(addr) post: walk.rtte.sh == SHAREABILITY_OUTER

ID	Condition
rtte_sh_coh	pre: <code>PaIsDelegableCohDevMem(addr)</code> post: <code>walk.rtte.sh == SHAREABILITY_INNER</code>

B4.3.4.4 Footprint

ID	Value
state	<code>GranuleAt(addr).state</code>
rtte	<code>RttEntryAt(RttAt(walk.rtt_addr), entry_idx)</code>

DRAFT

B4.3.5 RMI_DEV_MEM_UNMAP command

Unmaps device memory.

Issue Consider how teardown of DRAM mappings (via RMI_DATA_DESTROY) composes with teardown of device memory mappings (via RMI_DEV_MEM_UNMAP). In each case, the command returns the IPA of the next live entry - but it doesn't tell the caller whether this is DRAM or IO. How then can the caller know which of the two commands to call next, while still avoiding a (race-prone) call to RMI_RTT_READ_ENTRY?

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.5.1 Interface

B4.3.5.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000173
rd	X1	63:0	Address	PA of the RD which owns the target device memory Granule
ipa	X2	63:0	Address	IPA at which the Granule is mapped in the target Realm
level	X3	63:0	Int64	RTT level

B4.3.5.1.2 Context

The RMI_DEV_MEM_UNMAP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt(rd)	false	Realm
walk	RmmRttWalkResult	RttWalk(realm, ipa, level, RMM_RTT_TREE_PRIMARY)	false	RTT walk result
rtte_state_pre	RmmRttEntryState	walk.rtte.state	true	RTT entry state
entry_idx	UInt64	RttEntryIndex(ipa, walk.level)	false	RTTE index
walk_top	Address	RttSkipNonLiveEntries(RttAt(walk.rtt_addr), walk.level, ipa)	false	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.5.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
pa	X1	63:0	Address	PA of the device memory Granule which was unmapped

Name	Register	Bits	Type	Description
top	X2	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

The values of the `result` and `top` output values for different command outcomes are summarized in the following table.

Scenario	result	top	walk.rtte.state
ipa is mapped at the target level	RMI_SUCCESS	> ipa	Before execution: ASSIGNED_DEV After execution: UNASSIGNED
ipa is not mapped	(RMI_ERROR_RTT, <= level)	> ipa	UNASSIGNED
ipa is mapped at a lower level	(RMI_ERROR_RTT, < level)	== ipa	ASSIGNED_DEV
RTT walk was not performed, due to any other command failure	Another error code	0	Unknown

See also:

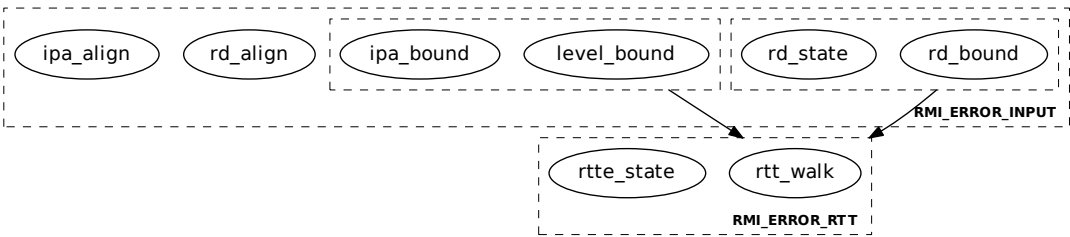
- [A5.6.8 RTTE liveness and RTT liveness](#)

B4.3.5.2 Failure conditions

ID	Condition
rd_align	pre: <code>!AddrIsGranuleAligned(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_bound	pre: <code>!PaIsDelegable(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_state	pre: <code>GranuleAt(rd).state != RD</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
level_bound	pre: <code>!RttLevelIsBlockOrPage(realm, level)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
ipa_align	pre: <code>!AddrIsRttLevelAligned(ipa, level)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
ipa_bound	pre: <code>!AddrIsProtected(ipa, realm)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rtt_walk	pre: <code>walk.level < level</code> post: <code>(ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))</code>
rtte_state	pre: <code>walk.rtte.state != ASSIGNED_DEV</code> post: <code>(ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))</code>

B4.3.5.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```

B4.3.5.3 Success conditions

ID	Condition
state	<code>GranuleAt(walk.rtte.addr).state == DELEGATED</code>
rtte_state	<code>walk.rtte.state == UNASSIGNED</code>
ripas_dev	pre: <code>walk.rtte.ripas == DEV</code> post: <code>walk.rtte.ripas == DESTROYED</code>
pa	<code>pa == walk.rtte.addr</code>
top	<code>top == walk_top</code>

B4.3.5.4 Footprint

ID	Value
state	<code>GranuleAt(walk.rtte.addr).state</code>
rtte	<code>RttEntryAt(RttAt(walk.rtt_addr), entry_idx)</code>

B4.3.6 RMI_FEATURES command

Read feature register.

The following table indicates which feature register is returned depending on the index provided.

Index	Feature register
0	RMI feature register 0
1	RMI feature register 1

See also:

- [Chapter A3 Feature discovery and configuration](#)

B4.3.6.1 Interface

B4.3.6.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000165
index	X1	63:0	UInt64	Feature register index

B4.3.6.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
value	X1	63:0	Bits64	Feature register value

B4.3.6.2 Failure conditions

The RMI_FEATURES command does not have any failure conditions.

B4.3.6.3 Success conditions

ID	Condition
value	<code>value == RmiFeatureRegisterEncode(index)</code>

B4.3.6.4 Footprint

The RMI_FEATURES command does not have any footprint.

B4.3.7 RMI_GRANULE_DELEGATE command

Delegates a Granule.

See also:

- [A2.2 Granule](#)
- [B4.3.8 RMI_GRANULE_UNDELEGATE command](#)
- [D1.2.1 Realm creation flow](#)

B4.3.7.1 Interface

B4.3.7.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000151
addr	X1	63:0	Address	PA of the target Granule

B4.3.7.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.7.2 Failure conditions

ID	Condition
gran_align	pre: !AddrIsGranuleAligned (addr) post: ResultEqual (result, RMI_ERROR_INPUT)
gran_bound	pre: !PaIsDelegable (addr) post: ResultEqual (result, RMI_ERROR_INPUT)
gran_state	pre: GranuleAt (addr).state != UNDELEGATED post: ResultEqual (result, RMI_ERROR_INPUT)

B4.3.7.2.1 Failure condition ordering

The RMI_GRANULE_DELEGATE command does not have any failure condition orderings.

B4.3.7.3 Success conditions

ID	Condition
gran_state	GranuleAt (addr).state == DELEGATED
gran_gpt	GranuleAt (addr).gpt == GPT_REALM

B4.3.7.4 Footprint

ID	Value
gran_gpt	<code>GranuleAt(addr).gpt</code>
gran_state	<code>GranuleAt(addr).state</code>

DRAFT

B4.3.8 RMI_GRANULE_UNDELEGATE command

Undelegates a Granule.

See also:

- [A2.2 Granule](#)
- [B4.3.7 RMI_GRANULE_DELEGATE command](#)
- [D1.2.5 Realm destruction flow](#)

B4.3.8.1 Interface

B4.3.8.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000152
addr	X1	63:0	Address	PA of the target Granule

B4.3.8.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.8.2 Failure conditions

ID	Condition
gran_align	pre: !AddrIsGranuleAligned (addr) post: ResultEqual (result, RMI_ERROR_INPUT)
gran_bound	pre: !PaIsDelegable (addr) post: ResultEqual (result, RMI_ERROR_INPUT)
gran_state	pre: GranuleAt (addr).state != DELEGATED post: ResultEqual (result, RMI_ERROR_INPUT)

B4.3.8.2.1 Failure condition ordering

The RMI_GRANULE_UNDELEGATE command does not have any failure condition orderings.

B4.3.8.3 Success conditions

ID	Condition
gran_gpt	GranuleAt (addr).gpt != GPT_REALM
gran_state	GranuleAt (addr).state == UNDELEGATED
gran_content	Contents of target Granule are wiped.

See also:

- [A2.2.5 Granule wiping](#)

B4.3.8.4 Footprint

ID	Value
gran_gpt	GranuleAt (addr) .gpt
gran_state	GranuleAt (addr) .state

DRAFT

B4.3.9 RMI_MEC_SET_PRIVATE command

Change state of a MEC to Private.

See also:

- [Chapter A11 Realm memory encryption](#)
- [B4.3.10 RMI_MEC_SET_SHARED command](#)

B4.3.9.1 Interface

B4.3.9.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400018D
mecid	X1	63:0	Bits64	MECID

B4.3.9.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.9.2 Failure conditions

ID	Condition
mecid_bound	pre: <code>UInt(mecid) > UInt(ImplFeatures().max_mecid)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
state	pre: <code>MecState(mecid) != MEC_STATE_SHARED</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
members	pre: <code>MecMembers(mecid) != 0</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>

B4.3.9.2.1 Failure condition ordering

The RMI_MEC_SET_PRIVATE command does not have any failure condition orderings.

B4.3.9.3 Success conditions

ID	Condition
mec_state	<code>MecState(mecid) == MEC_STATE_PRIVATE_UNASSIGNED</code>

B4.3.9.4 Footprint

The RMI_MEC_SET_PRIVATE command does not have any footprint.

B4.3.10 RMI_MEC_SET_SHARED command

Change state of a MEC to Shared.

See also:

- [Chapter A11 Realm memory encryption](#)
- [B4.3.9 RMI_MEC_SET_PRIVATE command](#)

B4.3.10.1 Interface

B4.3.10.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400018C
mecid	X1	63:0	Bits64	MECID

B4.3.10.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.10.2 Failure conditions

ID	Condition
mecid_bound	pre: <code>UInt(mecid) > UInt(ImplFeatures().max_mecid)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
state	pre: <code>MecState(mecid) != MEC_STATE_PRIVATE_UNASSIGNED</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>

B4.3.10.2.1 Failure condition ordering

The RMI_MEC_SET_SHARED command does not have any failure condition orderings.

B4.3.10.3 Success conditions

ID	Condition
mec_state	<code>MecState(mecid) == MEC_STATE_SHARED</code>

B4.3.10.4 Footprint

The RMI_MEC_SET_SHARED command does not have any footprint.

B4.3.11 RMI_PDEV_ABORT command

Abort device communication associated with a PDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.11.1 Interface

B4.3.11.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000174
pdev_ptr	X1	63:0	Address	PA of the PDEV

B4.3.11.1.2 Context

The RMI_PDEV_ABORT command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV
pdev_state_pre	RmmPdevState	pdev.state	true	Previous state

B4.3.11.1.3 Output values

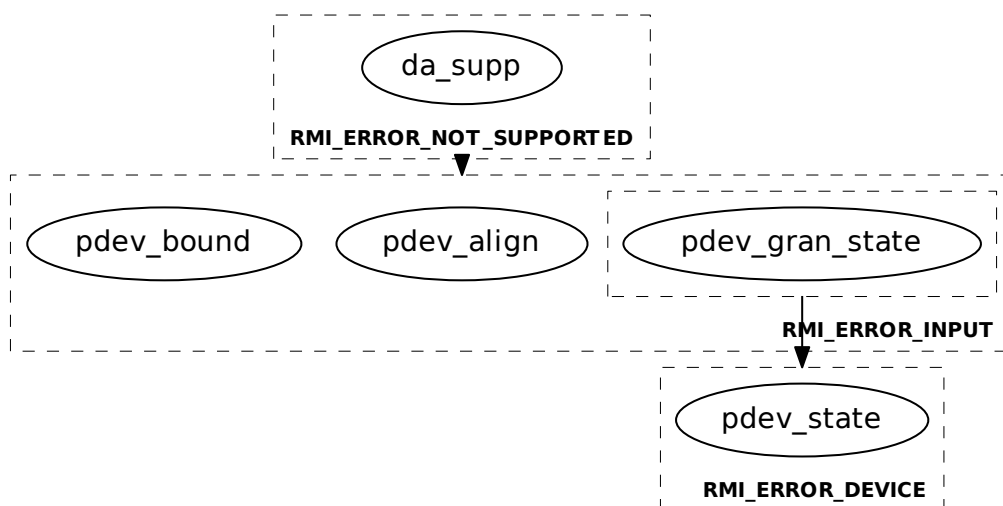
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.11.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegable (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt (pdev_ptr) .state != PDEV post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_state	pre: (pdev.state != PDEV_NEW && pdev.state != PDEV_HAS_KEY && pdev.state != PDEV_COMMUNICATING) post: ResultEqual (result, RMI_ERROR_DEVICE)

B4.3.11.2.1 Failure condition ordering

```
[da_supp] < [pdev_align, pdev_bound, pdev_gran_state]
[pdev_gran_state] < [pdev_state]
```



B4.3.11.3 Success conditions

ID	Condition
comm	pre: pdev_state_pre == PDEV_COMMUNICATING post: (pdev.state == PDEV_READY && pdev.comm_state == DEV_COMM_IDLE)
not_comm	pre: pdev_state_pre != PDEV_COMMUNICATING post: pdev.comm_state == DEV_COMM_PENDING

B4.3.11.4 Footprint

ID	Value
state	pdev.state
comm_state	pdev.comm_state

B4.3.12 RMI_PDEV_AUX_COUNT command

Get number of auxiliary Granules required for a PDEV.

B4.3.12.1 Interface

B4.3.12.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000156
flags	X1	63:0	Bits64	PDEV flags

B4.3.12.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
aux_count	X1	63:0	UInt64	Number of auxiliary Granules required for a PDEV

B4.3.12.2 Failure conditions

ID	Condition
da_supp	pre: <code>ImplFeatures().feat_da != FEATURE_TRUE</code> post: <code>ResultEqual(result, RMI_ERROR_NOT_SUPPORTED)</code>

B4.3.12.3 Success conditions

ID	Condition
aux_count	<code>aux_count == PdevAuxCount(RmiPdevFlagsDecode(flags))</code>

B4.3.12.4 Footprint

The RMI_PDEV_AUX_COUNT command does not have any footprint.

B4.3.13 RMI_PDEV_COMMUNICATE command

Perform device communication associated with a PDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.13.1 Interface

B4.3.13.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000175
pdev_ptr	X1	63:0	Address	PA of the PDEV
data_ptr	X2	63:0	Address	PA of the communication data structure

B4.3.13.1.2 Context

The RMI_PDEV_COMMUNICATE command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV
pdev_state_pre	RmmPdevState	PdevAt (pdev_ptr) .state	true	PDEV previous state
data	RmiDevCommData	RmiDevCommDataAt (data_ptr)	false	Device communication object

B4.3.13.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

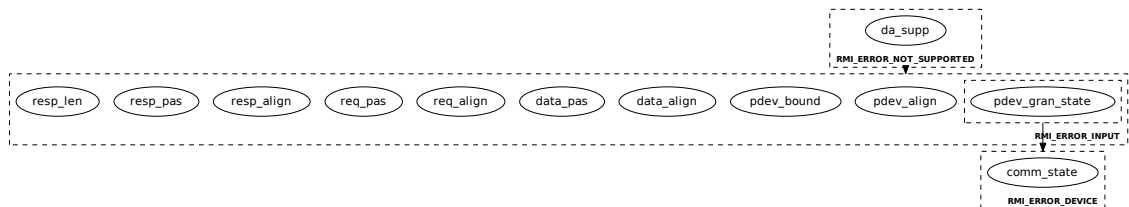
B4.3.13.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures ().feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegable (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt (pdev_ptr) .state != PDEV post: ResultEqual (result, RMI_ERROR_INPUT)
data_align	pre: !AddrIsGranuleAligned (data_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
data_pas	pre: !GranuleAccessPermitted(data_ptr, PAS_NS) post: ResultEqual(result, RMI_ERROR_INPUT)
req_align	pre: !AddrIsGranuleAligned(data.enter.req_addr) post: ResultEqual(result, RMI_ERROR_INPUT)
req_pas	pre: !GranuleAccessPermitted(data.enter.req_addr, PAS_NS) post: ResultEqual(result, RMI_ERROR_INPUT)
resp_align	pre: !AddrIsGranuleAligned(data.enter.resp_addr) post: ResultEqual(result, RMI_ERROR_INPUT)
resp_pas	pre: !GranuleAccessPermitted(data.enter.resp_addr, PAS_NS) post: ResultEqual(result, RMI_ERROR_INPUT)
resp_len	pre: data.enter.resp_len > RMM_Granule_Size post: ResultEqual(result, RMI_ERROR_INPUT)
comm_state	pre: (pdev.comm_state == DEV_COMM_IDLE pdev.comm_state == DEV_COMM_ERROR) post: ResultEqual(result, RMI_ERROR_DEVICE)

B4.3.13.2.1 Failure condition ordering

```
[da_supp] < [pdev_align, pdev_bound, pdev_gran_state, data_align,
             data_pas, req_align, req_pas, resp_align, resp_pas, resp_len]
[pdev_gran_state] < [comm_state]
```



B4.3.13.3 Success conditions

ID	Condition
comm_state	pdev.comm_state == DeviceCommunicate(pdev, data)
error	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_ERROR && pdev.state != PDEV_STOPPING) post: pdev.state == PDEV_ERROR
new	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev.state_pre == PDEV_NEW) post: pdev.state == PDEV_NEEDS_KEY
has_key	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev.state_pre == PDEV_HAS_KEY) post: pdev.state == PDEV_READY
ready	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev.state_pre == PDEV_READY) post: pdev.state == PDEV_READY

ID	Condition
stopped	pre: (DeviceCommunicate(pdev, data) != DEV_COMM_ACTIVE && pdev_state_pre == PDEV_STOPPING) post: pdev.state == PDEV_STOPPED
communicating	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev_state_pre == PDEV_COMMUNICATING) post: pdev.state == PDEV_READY
ide_resetting	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev_state_pre == PDEV_IDE_RESETTING) post: pdev.state == PDEV_READY

B4.3.13.4 Footprint

ID	Value
state	pdev.state
comm_state	pdev.comm_state

B4.3.14 RMI_PDEV_CREATE command

Create a PDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.14.1 Interface

B4.3.14.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000176
pdev_ptr	X1	63:0	Address	PA of the PDEV
params_ptr	X2	63:0	Address	PA of PDEV parameters

B4.3.14.1.2 Context

The RMI_PDEV_CREATE command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV
params	RmiPdevParams	RmiPdevParamsAt (params_ptr)	false	PDEV parameters

B4.3.14.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

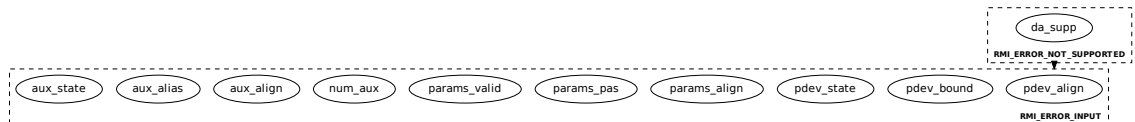
B4.3.14.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegableDram (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_state	pre: GranuleAt (pdev_ptr) .state != DELEGATED post: ResultEqual (result, RMI_ERROR_INPUT)
params_align	pre: !AddrIsGranuleAligned (params_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
params_pas	pre: !GranuleAccessPermitted (params_ptr, PAS_NS) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
params_valid	pre: <code>!RmiPdevParamsIsValid(params_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
num_aux	pre: <code>params.num_aux != PdevAuxCount(params.flags)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
aux_align	pre: <code>!AuxAligned32(params.aux, params.num_aux)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
aux_alias	pre: <code>AuxAlias32(pdev_ptr, params.aux, params.num_aux)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
aux_state	pre: <code>!AuxStateEqual32(params.aux, params.num_aux, DELEGATED)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>

B4.3.14.2.1 Failure condition ordering

```
[da_supp] < [pdev_align, pdev_bound, pdev_state, params_align,
             params_pas, params_valid, num_aux, aux_align, aux_alias,
             aux_state]
```



B4.3.14.3 Success conditions

ID	Condition
gran_state	<code>GranuleAt(pdev_ptr).state == PDEV</code>
pdev_id	<code>pdev.pdev_id == params.pdev_id</code>
prot	<code>Equal(pdev.prot, params.flags.prot)</code>
segment_id	<code>pdev.segment_id == params.segment_id</code>
ecam_addr	<code>pdev.ecam_addr == params.ecam_addr</code>
root_id	<code>pdev.root_id == params.root_id</code>
cert_id	<code>pdev.cert_id == params.cert_id</code>
rid_base	<code>pdev.rid_base == params.rid_base</code>
rid_top	<code>pdev.rid_top == params.rid_top</code>
hash_algo	<code>Equal(pdev.hash_algo, params.hash_algo)</code>
ide_sid	<code>pdev.ide_sid == params.ide_sid</code>
ncoh_num_addr_range	<code>pdev.ncoh_num_addr_range == params.ncoh_num_addr_range</code>
ncoh_addr_range	<code>RmiAddressRangesEqual16(pdev.ncoh_addr_range, params.ncoh_addr_range, params.ncoh_num_addr_range)</code>

ID	Condition
coh_num_addr_range	<code>pdev.coh_num_addr_range == params.coh_num_addr_range</code>
coh_addr_range	<code>RmiAddressRangesEqual4(pdev.coh_addr_range, params.coh_addr_range, params.coh_num_addr_range)</code>
state	<code>pdev.state == PDEV_NEW</code>
comm_state	<code>pdev.comm_state == DEV_COMM_PENDING</code>
num_vdevs	<code>pdev.num_vdevs == 0</code>
aux	<code>AuxEqual32(pdev.aux, params.aux, PdevAuxCount(params.flags))</code>
num_aux	<code>pdev.num_aux == PdevAuxCount(params.flags)</code>
aux_state	<code>AuxStateEqual32(pdev.aux, PdevAuxCount(params.flags), PDEV_AUX)</code>

B4.3.14.4 Footprint

ID	Value
state	<code>GranuleAt(pdev_ptr).state</code>
aux_state	<code>AuxStates(pdev.aux, PdevAuxCount(params.flags))</code>

B4.3.15 RMI_PDEV_DESTROY command

Destroy a PDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.15.1 Interface

B4.3.15.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000177
pdev_ptr	X1	63:0	Address	PA of the PDEV

B4.3.15.1.2 Context

The RMI_PDEV_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
pdev_pre	RmmPdev	PdevAt (pdev_ptr)	true	PDEV

B4.3.15.1.3 Output values

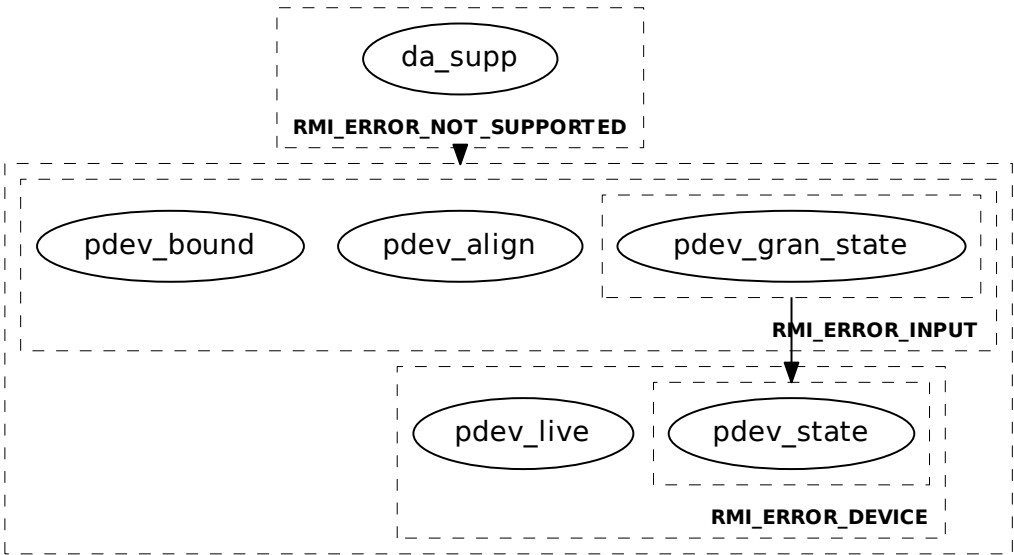
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.15.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegable (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt (pdev_ptr) .state != PDEV post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_state	pre: pdev_pre.state != PDEV_STOPPED post: ResultEqual (result, RMI_ERROR_DEVICE)
pdev_live	pre: pdev_pre.num_vdevs != 0 post: ResultEqual (result, RMI_ERROR_DEVICE)

B4.3.15.2.1 Failure condition ordering

[pdev_gran_state] < [pdev_state]
[da_supp] < [pdev_align, pdev_bound, pdev_gran_state, pdev_state,
pdev_live]



B4.3.15.3 Success conditions

ID	Condition
gran_state	<code>GranuleAt(pdev_ptr).state == DELEGATED</code>
aux_state	<code>AuxStateEqual32(pdev_pre.aux, pdev_pre.num_aux, DELEGATED)</code>

B4.3.15.4 Footprint

ID	Value
state	<code>GranuleAt(pdev_ptr).state</code>
aux_state	<code>AuxStates(pdev_pre.aux, prev_pre._num_aux)</code>

B4.3.16 RMI_PDEV_GET_STATE command

Get state of a PDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.16.1 Interface

B4.3.16.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000178
pdev_ptr	X1	63:0	Address	PA of the PDEV

B4.3.16.1.2 Context

The RMI_PDEV_GET_STATE command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV

B4.3.16.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
state	X1	7:0	RmiPdevState	PDEV state

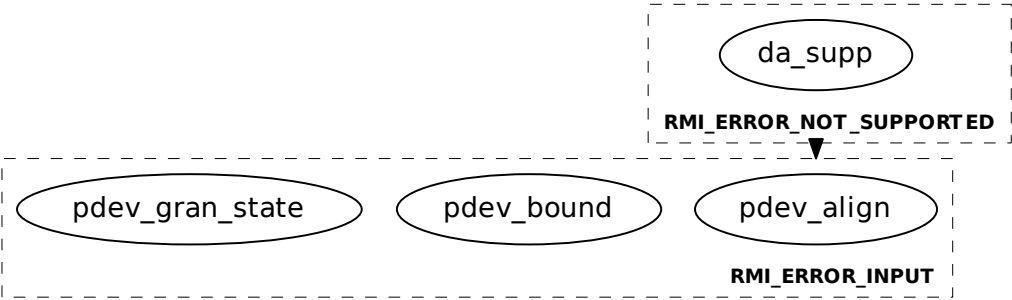
The following unused bits of RMI_PDEV_GET_STATE output values MBZ: X1[63:8].

B4.3.16.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegable (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt (pdev_ptr) .state != PDEV post: ResultEqual (result, RMI_ERROR_INPUT)

B4.3.16.2.1 Failure condition ordering

[da_supp] < [pdev_align, pdev_bound, pdev_gran_state]



B4.3.16.3 Success conditions

ID	Condition
state	<code>Equal(state, pdev.state)</code>

B4.3.16.4 Footprint

The RMI_PDEV_GET_STATE command does not have any footprint.

B4.3.17 RMI_PDEV_IDE_RESET command

Reset the IDE link of a PDEV.

B4.3.17.1 Interface

B4.3.17.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000179
pdev_ptr	X1	63:0	Address	PA of the PDEV

B4.3.17.1.2 Context

The RMI_PDEV_IDE_RESET command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV

B4.3.17.1.3 Output values

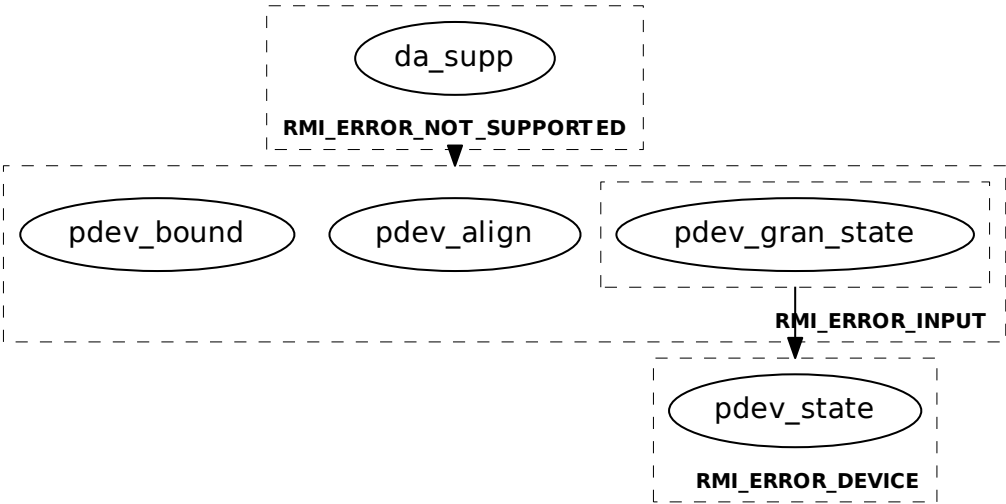
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.17.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures().feat_da != FEATURE_TRUE post: ResultEqual(result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned(pdev_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegable(pdev_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != PDEV post: ResultEqual(result, RMI_ERROR_INPUT)
pdev_state	pre: pdev.state != PDEV_READY post: ResultEqual(result, RMI_ERROR_DEVICE)

B4.3.17.2.1 Failure condition ordering

```
[da_supp] < [pdev_align, pdev_bound, pdev_gran_state]
[pdev_gran_state] < [pdev_state]
```



B4.3.17.3 Success conditions

ID	Condition
pdev_state	pdev.state == PDEV_IDE_RESETTING
comm_state	pdev.comm_state == DEV_COMM_PENDING

B4.3.17.4 Footprint

ID	Value
state	pdev.state
comm_state	pdev.comm_state

B4.3.18 RMI_PDEV_NOTIFY command

Notify the RMM of an event related to a PDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.18.1 Interface

B4.3.18.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400017A
pdev_ptr	X1	63:0	Address	PA of the PDEV
ev	X2	7:0	RmiPdevEvent	Event type

The following unused bits of RMI_PDEV_NOTIFY input values SBZ: X2[63:8].

B4.3.18.1.2 Context

The RMI_PDEV_NOTIFY command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV

B4.3.18.1.3 Output values

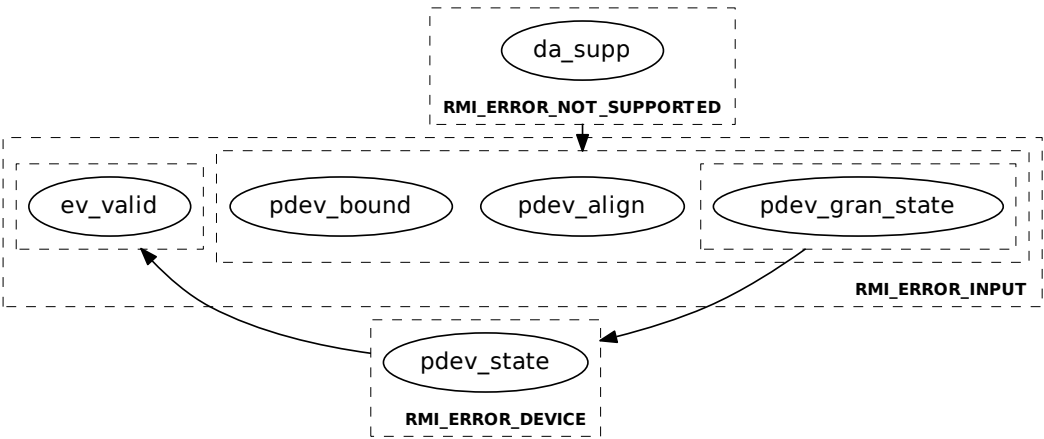
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.18.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegable (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt (pdev_ptr) .state != PDEV post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_state	pre: pdev.state != PDEV_READY post: ResultEqual (result, RMI_ERROR_DEVICE)
ev_valid	pre: !RmiPdevEventIsValid (ev) post: ResultEqual (result, RMI_ERROR_INPUT)

B4.3.18.2.1 Failure condition ordering

[da_supp] < [pdev_align, pdev_bound, pdev_gran_state]
[pdev_gran_state] < [pdev_state]
[pdev_state] < [ev_valid]



B4.3.18.3 Success conditions

ID	Condition
pdev_state	pdev.state == PDEV_COMMUNICATING
comm_state	pdev.comm_state == DEV_COMM_PENDING

B4.3.18.4 Footprint

ID	Value
state	pdev.state
comm_state	pdev.comm_state

B4.3.19 RMI_PDEV_SET_PUBKEY command

Provide public key associated with a PDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.19.1 Interface

B4.3.19.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400017B
pdev_ptr	X1	63:0	Address	PA of the PDEV
params_ptr	X2	63:0	Address	PA of the key parameters

B4.3.19.1.2 Context

The RMI_PDEV_SET_PUBKEY command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV
params	RmiPublicKeyParams	RmiPublicKeyParamsAt (↪params_ptr)	false	Public key parameters

B4.3.19.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

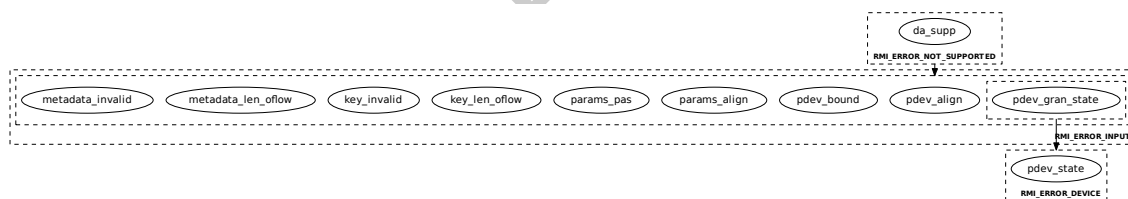
B4.3.19.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegable (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt (pdev_ptr) .state != PDEV post: ResultEqual (result, RMI_ERROR_INPUT)
params_align	pre: !AddrIsGranuleAligned (params_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
params_pas	pre: !GranuleAccessPermitted (params_ptr, PAS_NS) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
key_len_oflow	pre: params.key_len > 0x1000 post: ResultEqual(result, RMI_ERROR_INPUT)
metadata_len_oflow	pre: params.metadata_len > 0x1000 post: ResultEqual(result, RMI_ERROR_INPUT)
key_invalid	pre: Key is invalid, for example length is invalid for specified signature algorithm. post: ResultEqual(result, RMI_ERROR_INPUT)
metadata_invalid	pre: Metadata is invalid, for example length is invalid for specified signature algorithm. post: ResultEqual(result, RMI_ERROR_INPUT)
pdev_state	pre: pdev.state != PDEV_NEEDS_KEY post: ResultEqual(result, RMI_ERROR_DEVICE)

B4.3.19.2.1 Failure condition ordering

```
[da_supp] < [pdev_align, pdev_bound, pdev_gran_state, params_align,
             params_pas, key_len_oflow, key_invalid, metadata_len_oflow,
             metadata_invalid]
[pdev_gran_state] < [pdev_state]
```



B4.3.19.3 Success conditions

ID	Condition
state	pdev.state == PDEV_HAS_KEY
comm_state	pdev.comm_state == DEV_COMM_PENDING

B4.3.19.4 Footprint

ID	Value
state	pdev.state
comm_state	pdev.comm_state

B4.3.20 RMI_PDEV_STOP command

Stop a PDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.20.1 Interface

B4.3.20.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400017C
pdev_ptr	X1	63:0	Address	PA of the PDEV

B4.3.20.1.2 Context

The RMI_PDEV_STOP command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV

B4.3.20.1.3 Output values

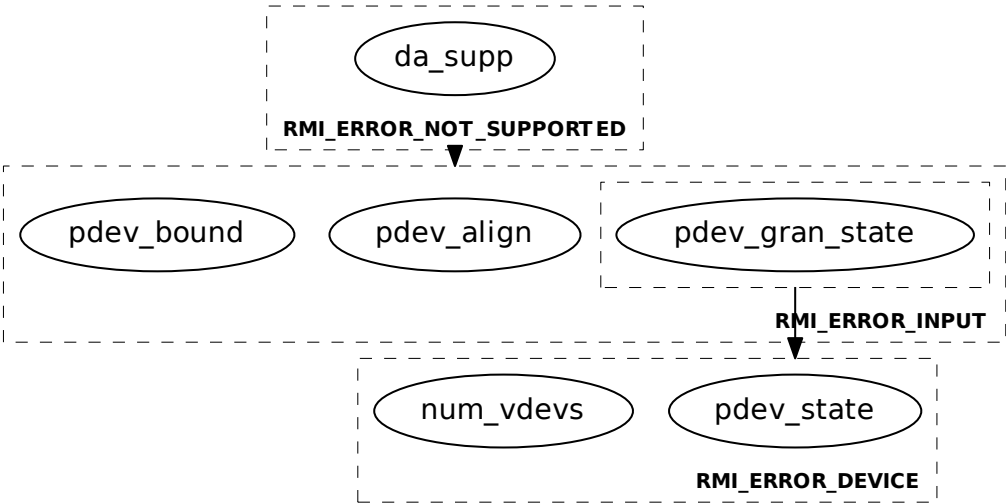
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.20.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_bound	pre: !PaIsDelegable (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt (pdev_ptr) .state != PDEV post: ResultEqual (result, RMI_ERROR_INPUT)
pdev_state	pre: (pdev.state == PDEV_COMMUNICATING pdev.state == PDEV_STOPPING pdev.state == PDEV_STOPPED) post: ResultEqual (result, RMI_ERROR_DEVICE)
num_vdevs	pre: pdev.num_vdevs != 0 post: ResultEqual (result, RMI_ERROR_DEVICE)

B4.3.20.2.1 Failure condition ordering

[da_supp] < [pdev_align, pdev_bound, pdev_gran_state]
[pdev_gran_state] < [pdev_state, num_vdevs]



B4.3.20.3 Success conditions

ID	Condition
pdev_state	pdev.state == PDEV_STOPPING
comm_state	pdev.comm_state == DEV_COMM_PENDING

B4.3.20.4 Footprint

ID	Value
state	pdev.state
comm_state	pdev.comm_state

B4.3.21 RMI_PSCI_COMPLETE command

Completes a pending PSCI command which was called with an MPIDR argument, by providing the corresponding REC.

See also:

- [A4.3.7 REC exit due to PSCI](#)
- [B6.3.1 PSCI_AFFINITY_INFO command](#)
- [B6.3.3 PSCI_CPU_ON command](#)
- [D1.4 PSCI flows](#)

B4.3.21.1 Interface

B4.3.21.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000164
calling_rec_ptr	X1	63:0	Address	PA of the calling REC
target_rec_ptr	X2	63:0	Address	PA of the target REC
status	X3	63:0	PsciReturnCode	Status of the PSCI request

B4.3.21.1.2 Context

The RMI_PSCI_COMPLETE command operates on the following context.

Name	Type	Value	Before	Description
calling_rec	RmmRec	RecAt (calling_rec_ptr)	false	Calling REC
target_rec	RmmRec	RecAt (target_rec_ptr)	false	Target REC

B4.3.21.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.21.2 Failure conditions

ID	Condition
alias	pre: calling_rec_ptr == target_rec_ptr post: ResultEqual (result, RMI_ERROR_INPUT)
calling_align	pre: !AddrIsGranuleAligned (calling_rec_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
calling_bound	pre: !PaIsDelegable (calling_rec_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
calling_state	pre: <code>GranuleAt(calling_rec_ptr).state != REC</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
target_align	pre: <code>!AddrIsGranuleAligned(target_rec_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
target_bound	pre: <code>!PaIsDelegable(target_rec_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
target_state	pre: <code>GranuleAt(target_rec_ptr).state != REC</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pending	pre: <code>calling_rec.pending != REC_PENDING_PSCI</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
owner	pre: <code>target_rec.owner != calling_rec.owner</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
target	pre: <code>target_rec.mpidr != calling_rec.gprs[1]</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
status	pre: <code>!PsciReturnCodePermitted(calling_rec, target_rec, status)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>

B4.3.21.2.1 Failure condition ordering

The RMI_PSCI_COMPLETE command does not have any failure condition orderings.

B4.3.21.3 Success conditions

ID	Condition
pending	<code>calling_rec.pending == REC_PENDING_NONE</code>
on_already	pre: <code>(status == PSCI_SUCCESS</code> <code>&& calling_rec.gprs[0] == FID_PSCI_CPU_ON</code> <code>&& target_rec.flags.runnable == RUNNABLE)</code> post: <code>(calling_rec.gprs[0] ==</code> <code>PsciReturnCodeEncode(PSCI_ALREADY_ON))</code>

ID	Condition
on_success	<pre> pre: (status == PSCI_SUCCESS && calling_rec.gprs[0] == FID_PSCI_CPU_ON && target_rec.flags.runnable != RUNNABLE) post: (target_rec.gprs[0] == calling_rec.gprs[3] && target_rec.gprs[1] == Zeros(64) && target_rec.gprs[2] == Zeros(64) && target_rec.gprs[3] == Zeros(64) && target_rec.gprs[4] == Zeros(64) && target_rec.gprs[5] == Zeros(64) && target_rec.gprs[6] == Zeros(64) && target_rec.gprs[7] == Zeros(64) && target_rec.gprs[8] == Zeros(64) && target_rec.gprs[9] == Zeros(64) && target_rec.gprs[10] == Zeros(64) && target_rec.gprs[11] == Zeros(64) && target_rec.gprs[12] == Zeros(64) && target_rec.gprs[13] == Zeros(64) && target_rec.gprs[14] == Zeros(64) && target_rec.gprs[15] == Zeros(64) && target_rec.gprs[16] == Zeros(64) && target_rec.gprs[17] == Zeros(64) && target_rec.gprs[18] == Zeros(64) && target_rec.gprs[19] == Zeros(64) && target_rec.gprs[20] == Zeros(64) && target_rec.gprs[21] == Zeros(64) && target_rec.gprs[22] == Zeros(64) && target_rec.gprs[23] == Zeros(64) && target_rec.gprs[24] == Zeros(64) && target_rec.gprs[25] == Zeros(64) && target_rec.gprs[26] == Zeros(64) && target_rec.gprs[27] == Zeros(64) && target_rec.gprs[28] == Zeros(64) && target_rec.gprs[29] == Zeros(64) && target_rec.gprs[30] == Zeros(64) && target_rec.gprs[31] == Zeros(64) && target_rec.pc == calling_rec.gprs[2] && target_rec.flags.runnable == RUNNABLE && calling_rec.gprs[0] == PsciReturnCodeEncode(PSCI_SUCCESS)) </pre>
affinity_on	<pre> pre: (status == PSCI_SUCCESS && calling_rec.gprs[0] == FID_PSCI_AFFINITY_INFO && target_rec.flags.runnable == RUNNABLE) post: (calling_rec.gprs[0] == PsciReturnCodeEncode(PSCI_SUCCESS)) </pre>
affinity_off	<pre> pre: (status == PSCI_SUCCESS && calling_rec.gprs[0] == FID_PSCI_AFFINITY_INFO && target_rec.flags.runnable != RUNNABLE) post: (calling_rec.gprs[0] == PsciReturnCodeEncode(PSCI_OFF)) </pre>
status	<pre> pre: status != PSCI_SUCCESS post: (calling_rec.gprs[0] == PsciReturnCodeEncode(status)) </pre>
args	<pre> (calling_rec.gprs[1] == Zeros(64) && calling_rec.gprs[2] == Zeros(64) && calling_rec.gprs[3] == Zeros(64)) </pre>

B4.3.21.4 Footprint

ID	Value
target_flags	target_rec.flags
target_gprs	target_rec.gprs
target_pc	target_rec.pc
calling_pend	calling_rec.pending
calling_gprs	calling_rec.gprs

DRAFT

B4.3.22 RMI_PSMMU_IRQ_NOTIFY command

Notify RMM of an SMMU interrupt.

See also:

- [A9.6.5 PSMMU interrupts](#)

B4.3.22.1 Interface

B4.3.22.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400016F
psmmu	X1	63:0	Address	PA of PSMMU
irq	X2	1:0	RmiSmmuIrq	SMMU IRQ

The following unused bits of RMI_PSMMU_IRQ_NOTIFY input values SBZ: X2[63:2].

B4.3.22.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
action	X1	0:0	RmiSmmuAction	Action required by Host
rd	X2	63:0	Address	PA of RD. This is valid if action == RMI_SMMU_ACTION_VIRQ.
vsmmu	X3	63:0	Address	PA of VSMMU. This is valid if action == RMI_SMMU_ACTION_VIRQ.
msi_addr	X4	63:0	Address	MSI address. This is valid if action == RMI_SMMU_ACTION_VIRQ.
msi_data	X5	63:0	Bits64	MSI data. This is valid if action == RMI_SMMU_ACTION_VIRQ.

The following unused bits of RMI_PSMMU_IRQ_NOTIFY output values MBZ: X1[63:1].

B4.3.22.2 Failure conditions

ID	Condition
psmmu_valid	pre: <code>!PsmmuAddrIsValid(psmmu)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>

B4.3.22.3 Success conditions

The RMI_PSMMU_IRQ_NOTIFY command does not have any success conditions.

B4.3.22.4 Footprint

The RMI_PSMMU_IRQ_NOTIFY command does not have any footprint.

DRAFT

B4.3.23 RMI_PSMMU_MSI_CONFIG command

Program the MSI configuration for the Realm side of the physical SMMU.

See also:

- [A9.6.5 PSMMU interrupts](#)

B4.3.23.1 Interface

B4.3.23.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400016E
psmmu	X1	63:0	Address	PA of PSMMU
gerr_addr	X2	63:0	Address	MSI address of the GERROR interrupt
gerr_data	X3	63:0	Bits64	MSI data of the GERROR interrupt
eventq_addr	X4	63:0	Address	MSI address of the EVENTQ interrupt
eventq_data	X5	63:0	Bits64	MSI data of the EVENTQ interrupt
priq_addr	X6	63:0	Address	MSI address of the PRIQ interrupt
priq_data	X7	63:0	Bits64	MSI data of the PRIQ interrupt

B4.3.23.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.23.2 Failure conditions

ID	Condition
psmmu_valid	pre: !PsmmuAddrIsValid (psmmu) post: ResultEqual (result, RMI_ERROR_INPUT)
psmmu_msi	pre: !PsmmuSupportsMsi (psmmu) post: ResultEqual (result, RMI_ERROR_INPUT)
gerror_valid	pre: !MsiAddrIsValid (gerr_addr) post: ResultEqual (result, RMI_ERROR_INPUT)
eventq_valid	pre: !MsiAddrIsValid (eventq_addr) post: ResultEqual (result, RMI_ERROR_INPUT)
priq_valid	pre: !MsiAddrIsValid (priq_addr) post: ResultEqual (result, RMI_ERROR_INPUT)

B4.3.23.2.1 Failure condition ordering

The RMI_PSMMU_MSI_CONFIG command does not have any failure condition orderings.

B4.3.23.3 Success conditions

ID	Condition
<code>gerr_addr</code>	<code>gerr_addr</code> is programmed to <code>SMMU_R_GERROR_IRQ_CFG0</code> with NS bit set.
<code>gerr_data</code>	<code>gerr_data</code> is programmed to <code>SMMU_R_GERROR_IRQ_CFG1</code> .
<code>eventq_addr</code>	<code>eventq_addr</code> is programmed to <code>SMMU_R_EVENTQ_IRQ_CFG0</code> with NS bit set.
<code>eventq_data</code>	<code>eventq_data</code> is programmed to <code>SMMU_R_EVENTQ_IRQ_CFG1</code> .
<code>priq_addr</code>	<code>priq_addr</code> is programmed to <code>SMMU_R_PRIQ_IRQ_CFG0</code> with NS bit set.
<code>priq_data</code>	<code>priq_data</code> is programmed to <code>SMMU_R_PRIQ_IRQ_CFG1</code> .

B4.3.23.4 Footprint

The `RMI_PSMMU_MSI_CONFIG` command does not have any footprint.

DRAFT

B4.3.24 RMI_REALM_ACTIVATE command

Activates a Realm.

See also:

- [A2.1 Realm](#)

B4.3.24.1 Interface

B4.3.24.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000157
rd	X1	63:0	Address	PA of the RD

B4.3.24.1.2 Context

The RMI_REALM_ACTIVATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm

B4.3.24.1.3 Output values

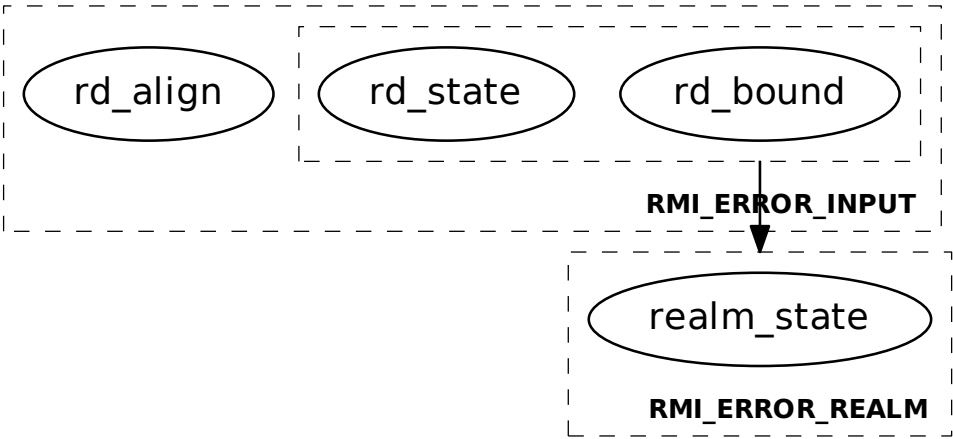
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.24.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)
realm_state	pre: realm.state != REALM_NEW post: ResultEqual (result, RMI_ERROR_REALM)

B4.3.24.2.1 Failure condition ordering

[rd_bound, rd_state] < [realm_state]



B4.3.24.3 Success conditions

ID	Condition
realm_state	realm.state == REALM_ACTIVE

B4.3.24.4 Footprint

ID	Value
realm_state	realm.state

B4.3.25 RMI_REALM_CREATE command

Creates a Realm.

See also:

- [A2.1 Realm](#)
- [A2.1.6 Realm parameters](#)
- [B4.3.26 RMI_REALM_DESTROY command](#)
- [D1.2.1 Realm creation flow](#)

B4.3.25.1 Interface

B4.3.25.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000158
rd	X1	63:0	Address	PA of the RD
params_ptr	X2	63:0	Address	PA of Realm parameters

B4.3.25.1.2 Context

The RMI_REALM_CREATE command operates on the following context.

Name	Type	Value	Before	Description
params	RmiRealmParams	RmiRealmParamsAt (params_ptr)	false	Realm parameters
realm	RmmRealm	RealmAt (rd)	false	Realm
mec_members_pre	UInt64	MecMembers (params.mecid)	true	Number of Realms which are members of the Realm's MEC
mec_state_pre	RmmMecState	MecState (params.mecid)	true	MECID state

B4.3.25.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.25.2 Failure conditions

ID	Condition
params_align	pre: !AddrIsGranuleAligned (params_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
params_pas	pre: !GranuleAccessPermitted (params_ptr, PAS_NS) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
params_valid	pre: <code>!RmiRealmParamsIsValid(params_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
params_supp	pre: <code>!RealmParamsSupported(params)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
alias	pre: <code>AddrInRange(rd, params.rtt_base, (params.rtt_num_start - 1) * RMM_GRANULE_SIZE)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_align	pre: <code>!AddrIsGranuleAligned(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_bound	pre: <code>!PaIsDelegableDram(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_state	pre: <code>GranuleAt(rd).state != DELEGATED</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rtt_align	pre: <code>!AddrIsAligned(params.rtt_base, params.rtt_num_start * RMM_GRANULE_SIZE)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rtt_num_level	pre: <code>!RttConfigIsValid(params.s2sz, params.rtt_level_start, params.rtt_num_start)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rtt_state	pre: <code>!RttsStateEqual(params.rtt_base, params.rtt_num_start, DELEGATED)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vmid_valid	pre: <code>(!VmidsAreValid(params.vmid, params.aux_vmid) !VmidsAreFree(params.vmid, params.aux_vmid))</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
mecid_bound	pre: <code>UInt(params.mecid) > UInt(ImplFeatures().max_mecid)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
mecid_state	pre: <code>MecState(params.mecid) == MEC_STATE_PRIVATE_ASSIGNED</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>

B4.3.25.2.1 Failure condition ordering

The RMI_REALM_CREATE command does not have any failure condition orderings.

B4.3.25.3 Success conditions

ID	Condition
rd_state	<code>GranuleAt(rd).state == RD</code>
realm_state	<code>realm.state == REALM_NEW</code>
rec_index	<code>realm.rec_index == 0</code>
rtt_base	<code>RealmRttBaseEqual(realm, params.rtt_base, params.aux_rtt_base)</code>
rtt_state	<code>RttsStateEqual(realm.rtt_base[0], realm.rtt_num_start, RTT)</code>

ID	Condition
rtte_p_states	<code>RttsAllProtectedEntriesState(realm.rtt_base[0], realm.rtt_num_start, UNASSIGNED)</code>
rtte_up_states	<code>RttsAllUnprotectedEntriesState(realm.rtt_base[0], realm.rtt_num_start, UNASSIGNED_NS)</code>
rtte_ripas	<code>RttsAllProtectedEntriesRipas(realm.rtt_base[0], realm.rtt_num_start, EMPTY)</code>
lpa2	<code>Equal(realm.feat_lpa2, params.flags0.lpa2)</code>
ipa_width	<code>realm.ipa_width == params.s2sz</code>
hash_algo	<code>Equal(realm.hash_algo, params.hash_algo)</code>
rim	<code>realm.measurements[0] == RimInit(realm.hash_algo, params)</code>
rem	<code>(realm.measurements[1] == Zeros(RMM_REALM_MEASUREMENT_WIDTH) && realm.measurements[2] == Zeros(RMM_REALM_MEASUREMENT_WIDTH) && realm.measurements[3] == Zeros(RMM_REALM_MEASUREMENT_WIDTH) && realm.measurements[4] == Zeros(RMM_REALM_MEASUREMENT_WIDTH))</code>
rtt_level	<code>realm.rtt_level_start == params.rtt_level_start</code>
rtt_num	<code>realm.rtt_num_start == params.rtt_num_start</code>
vmid	<code>RealmVmidEqual(realm, params.vmid, params.aux_vmid)</code>
rpv	<code>realm.rpv == params.rpv</code>
da	<code>Equal(realm.feat_da, params.flags0.da)</code>
rtt_tree_per_plane	<code>Equal(realm.rtt_tree_per_plane, params.flags1.rtt_tree_per_plane)</code>
num_aux_planes	<code>realm.num_aux_planes == params.num_aux_planes</code>
rtt_s2ap_encoding	<code>Equal(realm.rtt_s2ap_encoding, params.flags1.rtt_s2ap_encoding)</code>
lfa_policy	<code>Equal(realm.lfa_policy, params.flags0.lfa_policy)</code>
mecid	<code>realm.mecid == params.mecid</code>
mec_policy	<code>realm.mec_policy == MecPolicy(realm.mecid)</code>
mecid_private	<code>pre: mec_state_pre == MEC_STATE_PRIVATE_UNASSIGNED post: MecState(params.mecid) == MEC_STATE_PRIVATE_ASSIGNED</code>
mec_members	<code>pre: mec_state_pre == MEC_STATE_SHARED post: MecMembers(params.mecid) == mec_members_pre + 1</code>
num_recs	<code>realm.num_recs == 0</code>
num_vdevs	<code>realm.num_vdevs == 0</code>
vdev_count	<code>realm.vdev_count == 0</code>

B4.3.25.4 RMI_REALM_CREATE initialization of RIM

On successful execution of RMI_REALM_CREATE, the initial RIM value of the target Realm is calculated by the RMM as follows:

1. Allocate a zero-filled [RmiRealmParams](#) data structure to hold the measured Realm parameters.

2. Copy the following attributes from the Host-provided [RmiRealmParams](#) data structure into the measured Realm parameters data structure:
 - flags0
 - s2sz
 - sve_vl
 - num_bps
 - num_wps
 - pmu_num_ctrs
 - hash_algo
3. Using the RHA of the target Realm, compute the hash of the measured Realm parameters data structure. Set the RIM of the target Realm to this value, zero filling upper bytes if the RHA output is smaller than the size of the RIM.

See also:

- [A7.1.1 Realm Initial Measurement](#)
- [B3.79 RimInit function](#)
- [B4.4.29 RmiRealmParams type](#)

B4.3.25.5 Footprint

ID	Value
rd_state	GranuleAt (rd).state
rtt_state	RttsGranuleState (realm.rtt_base[0], realm.rtt_num_start)

B4.3.26 RMI_REALM_DESTROY command

Destroys a Realm.

See also:

- [A2.1 Realm](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [D1.2.5 Realm destruction flow](#)

B4.3.26.1 Interface

B4.3.26.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000159
rd	X1	63:0	Address	PA of the RD

B4.3.26.1.2 Context

The RMI_REALM_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
mec_members_pre	UInt64	MecMembers (realm_pre.mecid)	true	Number of Realms which are members of the Realm's MEC

B4.3.26.1.3 Output values

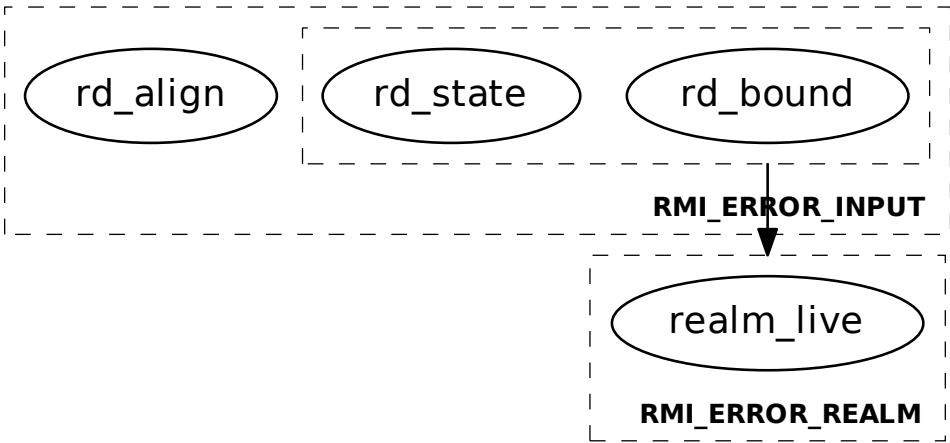
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.26.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)
realm_live	pre: RealmIsLive (rd) post: ResultEqual (result, RMI_ERROR_REALM)

B4.3.26.2.1 Failure condition ordering

[rd_bound, rd_state] < [realm_live]



B4.3.26.3 Success conditions

ID	Condition
rtt_state	<code>RttsStateEqual(realm_pre.rtt_base[0], realm_pre.rtt_num_start, DELEGATED)</code>
rd_state	<code>GranuleAt(rd).state == DELEGATED</code>
vmid	<code>VmidsAreFree(realm_pre.vmid)</code>
mecid_private	pre: <code>realm_pre.mec_policy == MEC_POLICY_PRIVATE</code> post: <code>MecState(realm_pre.mecid) == MEC_STATE_PRIVATE_UNASSIGNED</code>
mec_members	pre: <code>realm_pre.mec_policy == MEC_POLICY_SHARED</code> post: <code>MecMembers(realm_pre.mecid) == mec_members_pre - 1</code>

B4.3.26.4 Footprint

ID	Value
rd_state	<code>GranuleAt(rd).state</code>
rtt_state	<code>RttsGranuleState(realm_pre.rtt_base[0], realm_pre.rtt_num_start)</code>

B4.3.27 RMI_REC_AUX_COUNT command

Get number of auxiliary Granules required for a REC.

See also:

- [A2.3 Realm Execution Context](#)
- [B4.3.28 RMI_REC_CREATE command](#)
- [B4.4.36 RmiRecParams type](#)
- [D1.2.4 REC creation flow](#)

B4.3.27.1 Interface

B4.3.27.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000167
rd	X1	63:0	Address	PA of the RD for the target Realm

B4.3.27.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
aux_count	X1	63:0	UInt64	Number of auxiliary Granules required for a REC

B4.3.27.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)

B4.3.27.2.1 Failure condition ordering

The RMI_REC_AUX_COUNT command does not have any failure condition orderings.

B4.3.27.3 Success conditions

ID	Condition
aux_count	aux_count == RecAuxCount (rd)

B4.3.27.4 Footprint

The RMI_REC_AUX_COUNT command does not have any footprint.

B4.3.28 RMI_REC_CREATE command

Creates a REC.

See also:

- [A2.3 Realm Execution Context](#)
- [A2.3.3 REC index and MPIDR value](#)
- [B4.3.27 RMI_REC_AUX_COUNT command](#)
- [B4.3.29 RMI_REC_DESTROY command](#)
- [D1.2.4 REC creation flow](#)

B4.3.28.1 Interface

B4.3.28.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400015A
rd	X1	63:0	Address	PA of the RD for the target Realm
rec_ptr	X2	63:0	Address	PA of the target REC
params_ptr	X3	63:0	Address	PA of REC parameters

B4.3.28.1.2 Context

The RMI_REC_CREATE command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
realm	RmmRealm	RealmAt (rd)	false	Realm
params	RmiRecParams	RmiRecParamsAt (params_ptr)	false	REC parameters
rec	RmmRec	RecAt (rec_ptr)	false	REC

B4.3.28.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

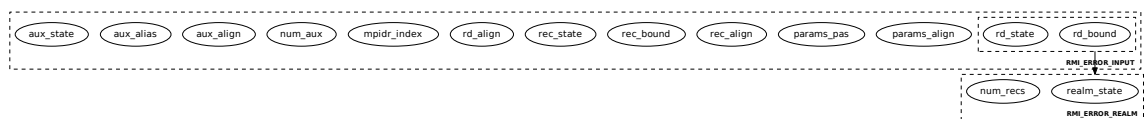
B4.3.28.2 Failure conditions

ID	Condition
params_align	pre: !AddrIsGranuleAligned (params_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
params_pas	pre: !GranuleAccessPermitted (params_ptr, PAS_NS) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
rec_align	pre: !AddrIsGranuleAligned(rec_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
rec_bound	pre: !PaIsDelegableDram(rec_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
rec_state	pre: GranuleAt(rec_ptr).state != DELEGATED post: ResultEqual(result, RMI_ERROR_INPUT)
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
realm_state	pre: realm_pre.state != REALM_NEW post: ResultEqual(result, RMI_ERROR_REALM)
num_recs	pre: realm_pre.num_recs == (2 ^ ImplFeatures().max_recs_order) - 1 post: ResultEqual(result, RMI_ERROR_REALM)
mpidr_index	pre: RecIndex(params.mpidr) != realm_pre.rec_index post: ResultEqual(result, RMI_ERROR_INPUT)
num_aux	pre: params.num_aux != RecAuxCount(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
aux_align	pre: !AuxAligned16(params.aux, params.num_aux) post: ResultEqual(result, RMI_ERROR_INPUT)
aux_alias	pre: AuxAlias16(rec_ptr, params.aux, params.num_aux) post: ResultEqual(result, RMI_ERROR_INPUT)
aux_state	pre: !AuxStateEqual16(params.aux, params.num_aux, DELEGATED) post: ResultEqual(result, RMI_ERROR_INPUT)

B4.3.28.2.1 Failure condition ordering

[rd_bound, rd_state] < [realm_state, num_recs]



B4.3.28.3 Success conditions

ID	Condition
rec_index	realm.rec_index == realm_pre.rec_index + 1
rec_gran_state	GranuleAt(rec_ptr).state == REC
rec_owner	rec.owner == rd
rec_attest	rec.attest_state == NO_ATTEST_IN_PROGRESS

ID	Condition
rec_mpidr	<code>MpidrEqual(rec.mpidr, params.mpidr)</code>
rec_state	<code>rec.state == REC_READY</code>
runnable	pre: <code>params.flags.runnable == RMI_RUNNABLE</code> post: <code>rec.flags.runnable == RUNNABLE</code>
not_runnable	pre: <code>params.flags.runnable == RMI_NOT_RUNNABLE</code> post: <code>rec.flags.runnable == NOT_RUNNABLE</code>
rec_gprs	<code>(rec.gprs[0] == params.gprs[0]</code> <code>&& rec.gprs[1] == params.gprs[1]</code> <code>&& rec.gprs[2] == params.gprs[2]</code> <code>&& rec.gprs[3] == params.gprs[3]</code> <code>&& rec.gprs[4] == params.gprs[4]</code> <code>&& rec.gprs[5] == params.gprs[5]</code> <code>&& rec.gprs[6] == params.gprs[6]</code> <code>&& rec.gprs[7] == params.gprs[7]</code> <code>&& rec.gprs[8] == Zeros(64)</code> <code>&& rec.gprs[9] == Zeros(64)</code> <code>&& rec.gprs[10] == Zeros(64)</code> <code>&& rec.gprs[11] == Zeros(64)</code> <code>&& rec.gprs[12] == Zeros(64)</code> <code>&& rec.gprs[13] == Zeros(64)</code> <code>&& rec.gprs[14] == Zeros(64)</code> <code>&& rec.gprs[15] == Zeros(64)</code> <code>&& rec.gprs[16] == Zeros(64)</code> <code>&& rec.gprs[17] == Zeros(64)</code> <code>&& rec.gprs[18] == Zeros(64)</code> <code>&& rec.gprs[19] == Zeros(64)</code> <code>&& rec.gprs[20] == Zeros(64)</code> <code>&& rec.gprs[21] == Zeros(64)</code> <code>&& rec.gprs[22] == Zeros(64)</code> <code>&& rec.gprs[23] == Zeros(64)</code> <code>&& rec.gprs[24] == Zeros(64)</code> <code>&& rec.gprs[25] == Zeros(64)</code> <code>&& rec.gprs[26] == Zeros(64)</code> <code>&& rec.gprs[27] == Zeros(64)</code> <code>&& rec.gprs[28] == Zeros(64)</code> <code>&& rec.gprs[29] == Zeros(64)</code> <code>&& rec.gprs[30] == Zeros(64)</code> <code>&& rec.gprs[31] == Zeros(64))</code>
rec_pc	<code>rec.pc == params.pc</code>
rim	pre: <code>params.flags.runnable == RMI_RUNNABLE</code> post: <code>realm.measurements[0] == RimExtendRec(</code> <code> realm_pre, params)</code>
rec_aux	<code>AuxEqual16(</code> <code> rec.aux, params.aux,</code> <code> RecAuxCount(rd))</code>
rec_aux_state	<code>AuxStateEqual16(</code> <code> rec.aux, RecAuxCount(rd), REC_AUX)</code>
ripas_addr	<code>rec.ripas_addr == Zeros(ADDRESS_WIDTH)</code>
ripas_top	<code>rec.ripas_top == Zeros(ADDRESS_WIDTH)</code>
pending	<code>rec.pending == REC_PENDING_NONE</code>
num_recs	<code>realm.num_recs == realm_pre.num_recs + 1</code>

ID	Condition
gic_owner	rec.gic_owner == 0

B4.3.28.4 RMI_REC_CREATE extension of RIM

On successful execution of RMI_REC_CREATE, if the new REC is runnable then the new RIM value of the target Realm is calculated by the RMM as follows:

1. Allocate a zero-filled [RmiRecParams](#) data structure to hold the measured REC parameters.
2. Copy the following attributes from the Host-provided [RmiRecParams](#) data structure into the measured REC parameters data structure:
 - gprs
 - pc
 - flags
3. Using the RHA of the target Realm, compute the hash of the measured REC parameters data structure.
4. Allocate an [RmmMeasurementDescriptorRec](#) data structure.
5. Populate the measurement descriptor:
 - Set the desc_type field to the descriptor type.
 - Set the len field to the descriptor length.
 - Set the rim field to the current RIM value of the target Realm.
 - Set the content field to the hash of the measured REC parameters.
6. Using the RHA of the target Realm, compute the hash of the measurement descriptor. Set the RIM of the target Realm to this value, zero filling upper bytes if the RHA output is smaller than the size of the RIM.

See also:

- [A7.1.1 Realm Initial Measurement](#)
- [B3.76 RimExtendRec function](#)
- [B4.4.36 RmiRecParams type](#)
- [C2.18 RmmMeasurementDescriptorRec type](#)

B4.3.28.5 Footprint

ID	Value
rec_index	realm.rec_index
rec_state	GranuleAt (rec).state
rec_aux_state	AuxStates (rec.aux, RecAuxCount (rd))
rim	realm.measurements[0]
num_recs	realm.num_recs

B4.3.29 RMI_REC_DESTROY command

Destroys a REC.

See also:

- [A2.3 Realm Execution Context](#)
- [B4.3.28 RMI_REC_CREATE command](#)
- [D1.2.5 Realm destruction flow](#)

B4.3.29.1 Interface

B4.3.29.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400015B
rec_ptr	X1	63:0	Address	PA of the target REC

B4.3.29.1.2 Context

The RMI_REC_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
rd_pre	Address	RecAt (rec_ptr).owner	true	RD address
realm_pre	RmmRealm	RealmAt (rd_pre)	true	Realm
realm	RmmRealm	RealmAt (rd_pre)	false	Realm
rec_pre	RmmRec	RecAt (rec_ptr)	true	REC
rec	RmmRec	RecAt (rec_ptr)	false	REC

B4.3.29.1.3 Output values

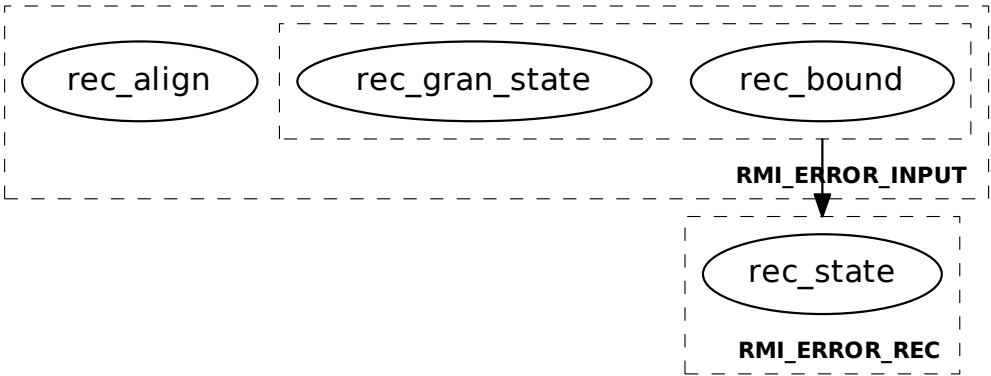
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.29.2 Failure conditions

ID	Condition
rec_align	pre: !AddrIsGranuleAligned (rec_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
rec_bound	pre: !PaIsDelegable (rec_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
rec_gran_state	pre: GranuleAt (rec_ptr).state != REC post: ResultEqual (result, RMI_ERROR_INPUT)
rec_state	pre: rec.state == REC_RUNNING post: ResultEqual (result, RMI_ERROR_REC)

B4.3.29.2.1 Failure condition ordering

$[rec_bound, rec_gran_state] < [rec_state]$



B4.3.29.3 Success conditions

ID	Condition
<code>rec_gran_state</code>	<code>GranuleAt(rec_ptr).state == DELEGATED</code>
<code>rec_aux_state</code>	<code>AuxStateEqual16(rec_pre.aux, RecAuxCount(rd_pre), DELEGATED)</code>
<code>num_recs</code>	<code>realm.num_recs == realm_pre.num_recs - 1</code>

B4.3.29.4 Footprint

ID	Value
<code>rec_state</code>	<code>GranuleAt(rec_ptr).state</code>
<code>rec_aux_state</code>	<code>AuxStates(rec_pre.aux, RecAuxCount(rd_pre))</code>
<code>num_recs</code>	<code>realm.num_recs</code>

B4.3.30 RMI_REC_ENTER command

Enter a REC.

See also:

- [A2.3 Realm Execution Context](#)
- [Chapter A4 Realm exception model](#)
- [D1.3.1 Realm entry and exit flow](#)

B4.3.30.1 Interface

B4.3.30.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400015C
rec_ptr	X1	63:0	Address	PA of the target REC
run_ptr	X2	63:0	Address	PA of RecRun object

The number of GICv3 List Register values which can be provided by the Host in RmiRecEnter, and which are returned in RmiRecExit, is reported by the RMI_FEATURES command.

See also:

- [A3.14 GICv3 virtualization](#)

B4.3.30.1.2 Context

The RMI_REC_ENTER command operates on the following context.

Name	Type	Value	Before	Description
run	RmiRecRun	RmiRecRunAt (run_ptr)	false	RecRun object
rec	RmmRec	RecAt (rec_ptr)	false	REC
realm	RmmRealm	RealmAt (rec.owner)	false	Realm

B4.3.30.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

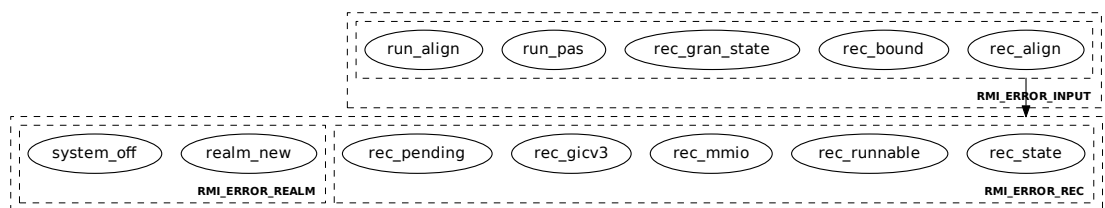
B4.3.30.2 Failure conditions

ID	Condition
run_align	pre: !AddrIsGranuleAligned (run_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
run_pas	pre: !GranuleAccessPermitted (run_ptr, PAS_NS) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
rec_align	pre: !AddrIsGranuleAligned(rec_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
rec_bound	pre: !PaIsDelegable(rec_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
rec_gran_state	pre: GranuleAt(rec_ptr).state != REC post: ResultEqual(result, RMI_ERROR_INPUT)
realm_new	pre: realm.state == REALM_NEW post: ResultEqual(result, RMI_ERROR_REALM, 0)
system_off	pre: realm.state == REALM_SYSTEM_OFF post: ResultEqual(result, RMI_ERROR_REALM, 1)
rec_state	pre: rec.state == REC_RUNNING post: ResultEqual(result, RMI_ERROR_REC)
rec_runnable	pre: rec.flags.runnable == NOT_RUNNABLE post: ResultEqual(result, RMI_ERROR_REC)
rec_mmio	pre: (run.enter.flags.emul_mmio == RMI_EMULATED_MMIO && rec.emulatable_abort != EMULATABLE_ABORT) post: ResultEqual(result, RMI_ERROR_REC)
rec_gicv3	pre: !Gicv3ConfigIsValid(run.enter.gicv3_hcr, run.enter.gicv3_lrs) post: ResultEqual(result, RMI_ERROR_REC)
rec_pending	pre: rec.pending != REC_PENDING_NONE post: ResultEqual(result, RMI_ERROR_REC)

B4.3.30.2.1 Failure condition ordering

```
[rec_align, rec_bound, rec_gran_state, run_pas, run_align] <
[rec_state, rec_runnable, rec_mmio, realm_new, system_off,
rec_gicv3, rec_pending]
```



B4.3.30.3 Success conditions

ID	Condition
rec_exit	run.exit contains Realm exit syndrome information.
rec_emul_abt	rec.emulatable_abort is updated.

B4.3.30.4 Footprint

ID	Value
emul_abt	rec.emulatable_abort

DRAFT

B4.3.31 RMI_RTT_AUX_CREATE command

Creates an auxiliary RTT.

See also:

- [A10.3.1 Auxiliary RTT](#)
- [B4.3.32 RMI_RTT_AUX_DESTROY command](#)
- [B4.3.33 RMI_RTT_AUX_FOLD command](#)

B4.3.31.1 Interface

B4.3.31.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400017D
rd	X1	63:0	Address	PA of the RD for the target Realm
rtt	X2	63:0	Address	PA of the target RTT
ipa	X3	63:0	Address	Base of the IPA range described by the RTT
level	X4	63:0	Int64	RTT level
index	X5	63:0	UInt64	RTT tree index

B4.3.31.1.2 Context

The RMI_RTT_AUX_CREATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level - 1, index)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
unfold	RmmRttEntry	RttWalk (realm, ipa, level - 1, index).rtte	true	RTTE before command execution

B4.3.31.1.3 Output values

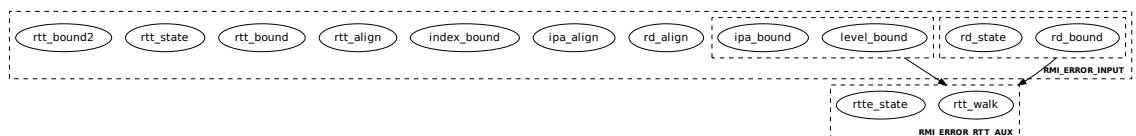
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.31.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
level_bound	pre: (!RttLevelIsValid(realm, level) RttLevelIsStarting(realm, level)) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsRttLevelAligned(ipa, level - 1) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: UInt(ipa) >= (2 ^ realm.ipa_width) post: ResultEqual(result, RMI_ERROR_INPUT)
index_bound	pre: (realm.rtt_tree_per_plane == FEATURE_FALSE index == RMM_RTT_TREE_PRIMARY index > realm.num_aux_planes) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_align	pre: !AddrIsGranuleAligned(rtt) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_bound	pre: !PaIsDelegableDram(rtt) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_state	pre: GranuleAt(rtt).state != DELEGATED post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_bound2	pre: ((realm.feat_lpa2 == FEATURE_FALSE) && (UInt(rtt) >= 2^48)) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < level - 1 post: ResultEqual(result, RMI_ERROR_RTT_AUX, walk.level)
rtte_state	pre: walk.rtte.state == TABLE post: ResultEqual(result, RMI_ERROR_RTT_AUX, walk.level)

B4.3.31.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.3.31.3 Success conditions

ID	Condition
rtt_state	<code>GranuleAt(rtt).state == RTT</code>
rtte_state	<code>walk.rtte.state == TABLE</code>
rtte_addr	<code>walk.rtte.addr == rtt</code>
rtte_c_ripas	pre: <code>AddrIsProtected(ipa, realm)</code> post: <code>RttAllEntriesRipas(RttAt(rtt), unfold.ripas)</code>
rtte_c_state	<code>RttAllEntriesState(RttAt(rtt), unfold.state)</code>
rtte_c_addr	pre: <code>(unfold.state != UNASSIGNED</code> <code>&& unfold.state != UNASSIGNED_NS)</code> post: <code>RttAllEntriesContiguous(RttAt(rtt), unfold.addr, level)</code>

B4.3.31.4 Footprint

ID	Value
rtt_state	<code>GranuleAt(rtt).state</code>
rtte	<code>RttEntry(walk.rtt_addr, entry_idx)</code>

B4.3.32 RMI_RTT_AUX_DESTROY command

Destroys an auxiliary RTT.

See also:

- [A10.3.1 Auxiliary RTT](#)
- [B4.3.31 RMI_RTT_AUX_CREATE command](#)
- [B4.3.33 RMI_RTT_AUX_FOLD command](#)

B4.3.32.1 Interface

B4.3.32.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400017E
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	Base of the IPA range described by the RTT
level	X3	63:0	Int64	RTT level
index	X4	63:0	UInt64	RTT tree index

B4.3.32.1.2 Context

The RMI_RTT_AUX_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level - 1, index)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
walk_top	Address	RttSkipNonLiveEntries (RttAt (walk.rtt_addr), walk.level, ipa)	false	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.32.1.3 Output values

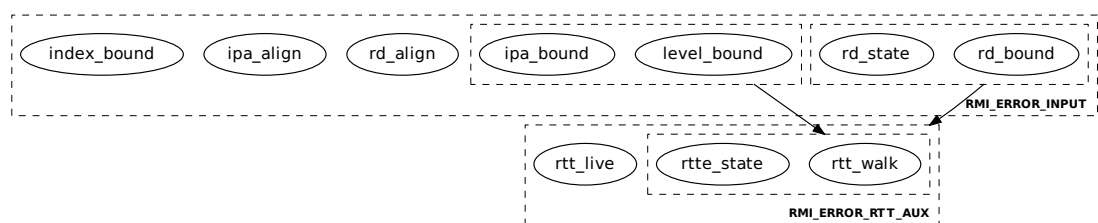
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
rtt	X1	63:0	Address	PA of the RTT which was destroyed
top	X2	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.32.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
level_bound	pre: (!RttLevelIsValid(realm, level) RttLevelIsStarting(realm, level)) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsRttLevelAligned(ipa, level - 1) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: UInt(ipa) >= (2 ^ realm.ipa_width) post: ResultEqual(result, RMI_ERROR_INPUT)
index_bound	pre: (realm.rtt_tree_per_plane == FEATURE_FALSE index == RMM_RTT_TREE_PRIMARY index > realm.num_aux_planes) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < level - 1 post: (ResultEqual(result, RMI_ERROR_RTT_AUX, walk.level) && (top == walk_top))
rtte_state	pre: walk.rtte.state != TABLE post: (ResultEqual(result, RMI_ERROR_RTT_AUX, walk.level) && (top == walk_top))
rtt_live	pre: RttIsLive(RttAt(walk.rtte.addr)) post: (ResultEqual(result, RMI_ERROR_RTT_AUX, level) && (top == ipa))

B4.3.32.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state, rtt_live]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.3.32.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == AUX_DESTROYED
ripas	walk.rtte.ripas == DESTROYED
rtt_state	GranuleAt(walk.rtte.addr).state == DELEGATED
rtt	rtt == walk.rtte.addr
top	top == walk_top

B4.3.32.4 Footprint

ID	Value
rtt_state	GranuleAt(walk.rtte.addr).state
rtte	RttEntry(walk.rtt_addr, entry_idx)

DRAFT

B4.3.33 RMI_RTT_AUX_FOLD command

Destroys a homogeneous auxiliary RTT.

See also:

- [A5.6.6 RTT folding](#)
- [A10.3.1 Auxiliary RTT](#)
- [B4.3.31 RMI_RTT_AUX_CREATE command](#)
- [B4.3.32 RMI_RTT_AUX_DESTROY command](#)

B4.3.33.1 Interface

B4.3.33.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400017F
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	Base of the IPA range described by the RTT
level	X3	63:0	Int64	RTT level
index	X4	63:0	UInt64	RTT tree index

B4.3.33.1.2 Context

The RMI_RTT_AUX_FOLD command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level - 1, index)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
fold_pre	RmmRttEntry	RttFold (RttAt (walk.rtte.addr))	true	Result of folding RTT

B4.3.33.1.3 Output values

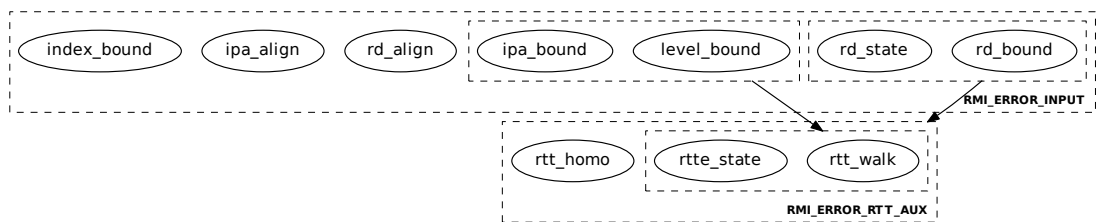
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
rtt	X1	63:0	Address	PA of the RTT which was destroyed

B4.3.33.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
level_bound	pre: (!RttLevelIsValid(realm, level) RttLevelIsStarting(realm, level)) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsRttLevelAligned(ipa, level - 1) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: UInt(ipa) >= (2 ^ realm.ipa_width) post: ResultEqual(result, RMI_ERROR_INPUT)
index_bound	pre: (realm.rtt_tree_per_plane == FEATURE_FALSE index == RMM_RTT_TREE_PRIMARY index > realm.num_aux_planes) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < level - 1 post: ResultEqual(result, RMI_ERROR_RTT_AUX, walk.level)
rtte_state	pre: walk.rtte.state != TABLE post: ResultEqual(result, RMI_ERROR_RTT_AUX, walk.level)
rtt_homo	pre: !RttIsHomogeneous(RttAt(walk.rtte.addr)) post: ResultEqual(result, RMI_ERROR_RTT_AUX, level)

B4.3.33.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state, rtt_homo]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.3.33.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == fold_pre.state
rtte_addr	pre: (fold_pre.state != UNASSIGNED && fold_pre.state != UNASSIGNED_NS) post: walk.rtte.addr == fold_pre.addr

ID	Condition
rtte_attr_prot	pre: fold_pre.state == ASSIGNED post: (RttMemAttrEqual(walk.rtte, fold_pre, RTT_PROTECTED) && RttS2APEqual(walk.rtte, fold_pre, S2AP_INDIRECT))
rtte_attr_unprot	pre: fold_pre.state == ASSIGNED_NS post: (RttMemAttrEqual(walk.rtte, fold_pre, RTT_UNPROTECTED) && RttS2APEqual(walk.rtte, fold_pre, realm.rtt_s2ap_encoding))
rtte_ripas	pre: AddrIsProtected(ipa, realm) post: walk.rtte.ripas == fold_pre.ripas
rtt_state	GranuleAt(walk.rtte.addr).state == DELEGATED
rtt	rtt == walk.rtte.addr

B4.3.33.4 Footprint

ID	Value
rtt_state	GranuleAt(walk.rtte.addr).state
rtte	RttEntry(walk.rtt_addr, entry_idx)

B4.3.34 RMI_RTT_AUX_MAP_PROTECTED command

Creates a mapping from an Protected IPA in an auxiliary RTT.

See also:

- [A10.3.1 Auxiliary RTT](#)
- [B4.3.36 RMI_RTT_AUX_UNMAP_PROTECTED command](#)

B4.3.34.1 Interface

B4.3.34.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000180
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	IPA in the target Realm
index	X3	63:0	UInt64	RTT tree index

B4.3.34.1.2 Context

The RMI_RTT_AUX_MAP_PROTECTED command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk_pri	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	Primary RTT walk result
walk_aux	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , index)	false	Auxiliary RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk_aux.level)	false	RTTE index

B4.3.34.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
fail_index	X1	63:0	UInt64	Index of RTT tree whose contents caused command to fail with RMI_ERROR_RTT
level_pri	X2	63:0	Int64	Level of RTTE reached by walk of primary RTT tree

Name	Register	Bits	Type	Description
state	X3	7:0	RmiRttEntryState	State of RTT entry whose contents caused command to fail with RMI_ERROR_RTT
ripas	X4	7:0	RmiRipas	RIPAS of RTT entry which caused command to fail with RMI_ERROR_RTT

The following unused bits of RMI_RTT_AUX_MAP_PROTECTED output values MBZ: X3[63:8], X4[63:8].

B4.3.34.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsGranuleAligned (ipa) post: ResultEqual (result, RMI_ERROR_INPUT)
ipa_bound	pre: !AddrIsProtected (ipa, realm) post: ResultEqual (result, RMI_ERROR_INPUT)
index_bound	pre: (realm.rtt_tree_per_plane == FEATURE_FALSE index == RMM_RTT_TREE_PRIMARY index > realm.num_aux_planes) post: ResultEqual (result, RMI_ERROR_INPUT)
pri_unassigned	pre: (walk_pri.rtte.state != ASSIGNED && walk_pri.rtte.state != ASSIGNED_DEV) post: (ResultEqual (result, RMI_ERROR_RTT , walk_pri.level) && (fail_index == RMM_RTT_TREE_PRIMARY) && (level_pri == walk_pri.level) && (state == RttEntryStateToRmi (walk_pri.rtte.state)) && (ripas == RipasToRmi (walk_pri.rtte.ripas)))
pri_ram	pre: (walk_pri.rtte.state == ASSIGNED && walk_pri.rtte.ripas != RAM) post: (ResultEqual (result, RMI_ERROR_RTT , walk_pri.level) && (fail_index == RMM_RTT_TREE_PRIMARY) && (level_pri == walk_pri.level) && (state == RttEntryStateToRmi (walk_pri.rtte.state)) && (ripas == RipasToRmi (walk_pri.rtte.ripas)))

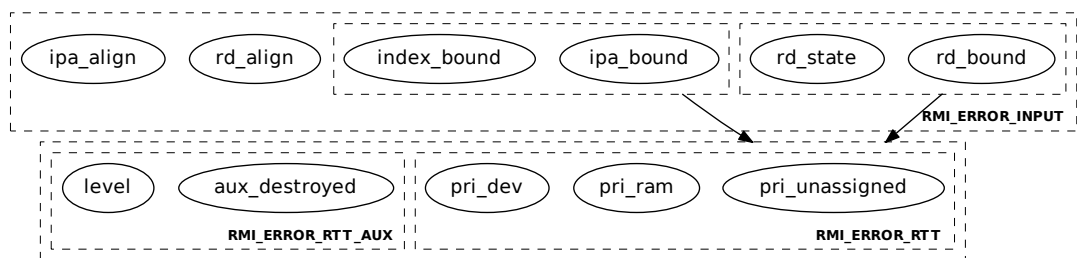
ID	Condition
pri_dev	<pre> pre: (walk_pri.rtte.state == ASSIGNED_DEV && walk_pri.rtte.ripas != DEV) post: (ResultEqual(result, RMI_ERROR_RTT, walk_pri.level) && (fail_index == RMM_RTT_TREE_PRIMARY) && (level_pri == walk_pri.level) && (state == RttEntryStateToRmi(walk_pri.rtte.state)) && (ripas == RipasToRmi(walk_pri.rtte.ripas))) </pre>
aux_destroyed	<pre> pre: walk_aux.rtte.state == AUX_DESTROYED post: (ResultEqual(result, RMI_ERROR_RTT_AUX, walk_aux.level) && (fail_index == index) && (level_pri == walk_pri.level) && (state == RttEntryStateToRmi(walk_aux.rtte.state)) && (ripas == RipasToRmi(walk_pri.rtte.ripas))) </pre>
level	<pre> pre: walk_aux.level < walk_pri.level post: (ResultEqual(result, RMI_ERROR_RTT_AUX, walk_aux.level) && (fail_index == index) && (level_pri == walk_pri.level) && (state == RttEntryStateToRmi(walk_aux.rtte.state)) && (ripas == RipasToRmi(walk_pri.rtte.ripas))) </pre>

B4.3.34.2.1 Failure condition ordering

```

[rd_bound, rd_state] < [pri_unassigned, pri_ram, pri_dev,
  aux_destroyed, level]
[ipa_bound, index_bound] < [pri_unassigned, pri_ram, pri_dev,
  aux_destroyed, level]

```



B4.3.34.3 Success conditions

ID	Condition
rtte_state	walk_aux.rtte.state == ASSIGNED
rtte_attr	walk_aux.rtte.attr_prot == walk_pri.rtte.attr_prot

ID	Condition
rtte_sh	walk_aux.rtte.sh == walk_pri.rtte.sh
rtte_addr	walk_aux.rtte.addr == walk_pri.rtte.addr + (entry_idx * RttLevelSize (walk_aux.level))

B4.3.34.4 Footprint

ID	Value
rtte	RttEntry(walk_aux.rtt_addr, entry_idx)

DRAFT

B4.3.35 RMI_RTT_AUX_MAP_UNPROTECTED command

Creates a mapping from an Unprotected IPA in an auxiliary RTT.

See also:

- [A10.3.1 Auxiliary RTT](#)
- [B4.3.37 RMI_RTT_AUX_UNMAP_UNPROTECTED command](#)

B4.3.35.1 Interface

B4.3.35.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000181
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	IPA in the target Realm
index	X3	63:0	UInt64	RTT tree index

B4.3.35.1.2 Context

The RMI_RTT_AUX_MAP_UNPROTECTED command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk_pri	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	Primary RTT walk result
walk_aux	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , index)	false	Auxiliary RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk_aux.level)	false	RTTE index

B4.3.35.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
fail_index	X1	63:0	UInt64	Index of RTT tree whose contents caused command to fail with RMI_ERROR_RTT
level_pri	X2	63:0	Int64	Level of RTTE reached by walk of primary RTT tree

Name	Register	Bits	Type	Description
state	X3	7:0	RmiRttEntryState	State of RTT entry whose contents caused command to fail with RMI_ERROR_RTT

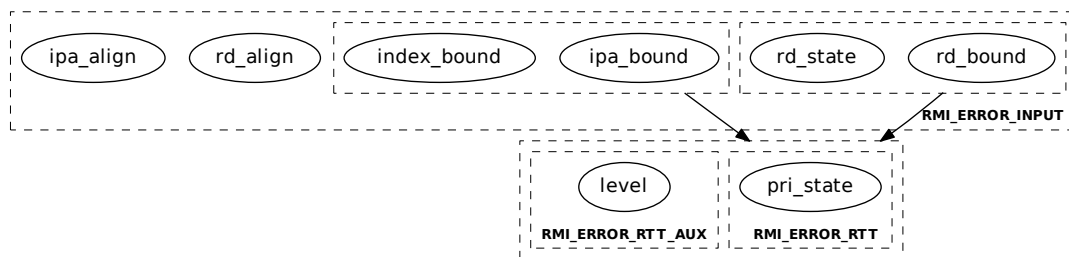
The following unused bits of RMI_RTT_AUX_MAP_UNPROTECTED output values MBZ: X3[63:8].

B4.3.35.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsGranuleAligned (ipa) post: ResultEqual (result, RMI_ERROR_INPUT)
ipa_bound	pre: (UInt(ipa) >= (2 ^ realm.ipa_width) AddrIsProtected (ipa, realm)) post: ResultEqual (result, RMI_ERROR_INPUT)
index_bound	pre: (realm.rtt_tree_per_plane == FEATURE_FALSE index == RMM_RTT_TREE_PRIMARY index > realm.num_aux_planes) post: ResultEqual (result, RMI_ERROR_INPUT)
pri_state	pre: walk_pri.rtte.state != ASSIGNED_NS post: (ResultEqual (result, RMI_ERROR_RTT , walk_pri.level) && (fail_index == RMM_RTT_TREE_PRIMARY) && (state == RttEntryStateToRmi (walk_pri.rtte.state)) && (level_pri == walk_pri.level))
level	pre: walk_aux.level < walk_pri.level post: (ResultEqual (result, RMI_ERROR_RTT_AUX , walk_aux.level) && (fail_index == index) && (level_pri == walk_pri.level) && (state == RttEntryStateToRmi (walk_aux.rtte.state)))

B4.3.35.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [pri_state, level]
[ipa_bound, index_bound] < [pri_state, level]
```



B4.3.35.3 Success conditions

ID	Condition
rtte_state	<code>walk_aux.rtte.state == ASSIGNED_NS</code>
rtte_attr_prot	<pre> pre: AddrIsProtected(ipa, realm) post: (RttMemAttrEqual(walk_aux.rtte, walk_pri.rtte, RTT_PROTECTED) && RttS2APEqual(walk_aux.rtte, walk_pri.rtte, realm.rtt_s2ap_encoding)) </pre>
rtte_attr_unprot	<pre> pre: !AddrIsProtected(ipa, realm) post: (RttMemAttrEqual(walk_aux.rtte, walk_pri.rtte, RTT_UNPROTECTED) && RttS2APEqual(walk_aux.rtte, walk_pri.rtte, realm.rtt_s2ap_encoding)) </pre>
rtte_addr	<pre> walk_aux.rtte.addr == walk_pri.rtte.addr + (entry_idx * RttLevelSize(walk_aux.level)) </pre>

B4.3.35.4 Footprint

ID	Value
rtte	<code>RttEntry(walk_aux.rtt_addr, entry_idx)</code>

B4.3.36 RMI_RTT_AUX_UNMAP_PROTECTED command

Removes a mapping from an Protected IPA in an auxiliary RTT.

See also:

- [A10.3.1 Auxiliary RTT](#)
- [B4.3.34 RMI_RTT_AUX_MAP_PROTECTED command](#)

B4.3.36.1 Interface

B4.3.36.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000183
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	IPA in the target Realm
index	X3	63:0	UInt64	RTT tree index

B4.3.36.1.2 Context

The RMI_RTT_AUX_UNMAP_PROTECTED command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL, index)	false	Auxiliary RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
walk_top	Address	RttSkipNonLiveEntries (RttAt (walk.rtt_addr), walk.level, ipa)	false	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.36.1.3 Output values

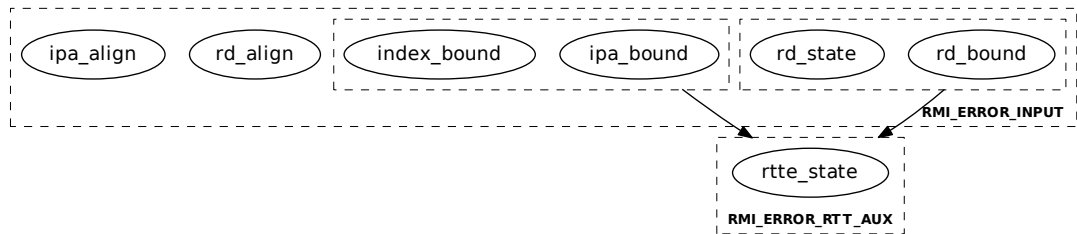
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
top	X1	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated
level	X2	63:0	Int64	RTT level reached by the RTT walk

B4.3.36.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsGranuleAligned(ipa) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: !AddrIsProtected(ipa, realm) post: ResultEqual(result, RMI_ERROR_INPUT)
index_bound	pre: (realm.rtt_tree_per_plane == FEATURE_FALSE index == RMM_RTT_TREE_PRIMARY index > realm.num_aux_planes) post: ResultEqual(result, RMI_ERROR_INPUT)
rtte_state	pre: walk.rtte.state != ASSIGNED post: (ResultEqual(result, RMI_ERROR_RTT_AUX, walk.level) && (top == walk_top))

B4.3.36.2.1 Failure condition ordering

[rd_bound, rd_state] < [rtte_state]
[ipa_bound, index_bound] < [rtte_state]



B4.3.36.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == UNASSIGNED
top	top == walk_top
level	level == walk.level

B4.3.36.4 Footprint

ID	Value
rtte	<code>RttEntry(walk.rtt_addr, entry_idx)</code>

DRAFT

B4.3.37 RMI_RTT_AUX_UNMAP_UNPROTECTED command

Removes a mapping from an Unprotected IPA in an auxiliary RTT.

See also:

- [A10.3.1 Auxiliary RTT](#)
- [B4.3.34 RMI_RTT_AUX_MAP_PROTECTED command](#)

B4.3.37.1 Interface

B4.3.37.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000184
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	IPA in the target Realm
index	X3	63:0	UInt64	RTT tree index

B4.3.37.1.2 Context

The RMI_RTT_AUX_UNMAP_UNPROTECTED command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , index)	false	Auxiliary RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
walk_top	Address	RttSkipNonLiveEntries (RttAt (walk.rtt_addr), walk.level, ipa)	false	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.37.1.3 Output values

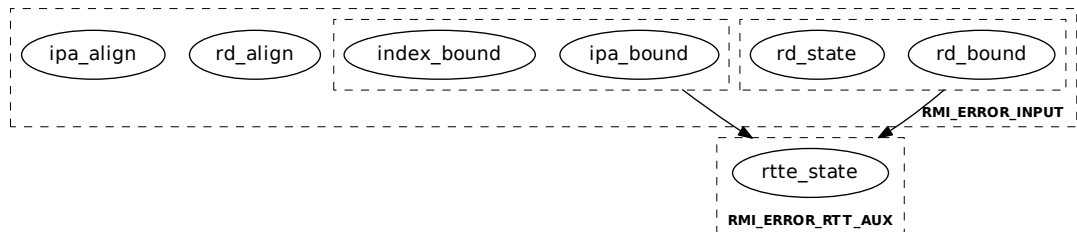
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
top	X1	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated
level	X2	63:0	Int64	RTT level reached by the RTT walk

B4.3.37.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsGranuleAligned(ipa) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: (UInt(ipa) >= (2 ^ realm.ipa_width) AddrIsProtected(ipa, realm)) post: ResultEqual(result, RMI_ERROR_INPUT)
index_bound	pre: (realm.rtt_tree_per_plane == FEATURE_FALSE index == RMM_RTT_TREE_PRIMARY index > realm.num_aux_planes) post: ResultEqual(result, RMI_ERROR_INPUT)
rtte_state	pre: walk.rtte.state != ASSIGNED_NS post: (ResultEqual(result, RMI_ERROR_RTT_AUX, walk.level) && (top == walk_top))

B4.3.37.2.1 Failure condition ordering

[rd_bound, rd_state] < [rtte_state]
[ipa_bound, index_bound] < [rtte_state]



B4.3.37.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == UNASSIGNED_NS
top	top == walk_top
level	level == walk.level

B4.3.37.4 Footprint

ID	Value
rtte	<code>RttEntry(walk.rtt_addr, entry_idx)</code>

DRAFT

B4.3.38 RMI_RTT_CREATE command

Creates a primary RTT.

See also:

- [A5.6 Realm Translation Table](#)
- [A5.6.7 RTT unfolding](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)
- [B4.3.41 RMI_RTT_FOLD command](#)

B4.3.38.1 Interface

B4.3.38.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400015D
rd	X1	63:0	Address	PA of the RD for the target Realm
rtt	X2	63:0	Address	PA of the target RTT
ipa	X3	63:0	Address	Base of the IPA range described by the RTT
level	X4	63:0	Int64	RTT level

B4.3.38.1.2 Context

The RMI_RTT_CREATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level - 1, RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
walk_pre	RmmRttWalkResult	RttWalk (realm, ipa, level - 1, RMM_RTT_TREE_PRIMARY)	true	RTT walk result before command execution
rtte_pre	RmmRttEntry	walk_pre.rtte	true	RTTE before command execution

B4.3.38.1.3 Output values

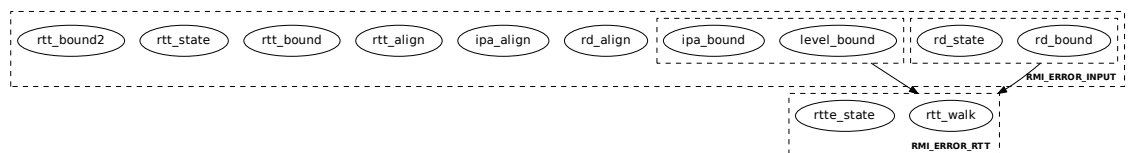
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.38.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
level_bound	pre: (!RttLevelIsValid(realm, level) RttLevelIsStarting(realm, level)) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsRttLevelAligned(ipa, level - 1) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: UInt(ipa) >= (2 ^ realm.ipa_width) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_align	pre: !AddrIsGranuleAligned(rtt) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_bound	pre: !PaIsDelegableDram(rtt) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_state	pre: GranuleAt(rtt).state != DELEGATED post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_bound2	pre: ((realm.feat_lpa2 == FEATURE_FALSE) && (UInt(rtt) >= 2^48)) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < level - 1 post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_state	pre: walk.rtte.state == TABLE post: ResultEqual(result, RMI_ERROR_RTT, walk.level)

B4.3.38.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.3.38.3 Success conditions

ID	Condition
rtt_state	GranuleAt(rtt).state == RTT
rtte_state	walk.rtte.state == TABLE

ID	Condition
rtte_addr	<code>walk.rtte.addr == rtt</code>
rtte_c_ripas	pre: <code>AddrIsProtected(ipa, realm)</code> post: <code>RttAllEntriesRipas(RttAt(rtt), rtte_pre.ripas)</code>
rtte_c_state	<code>RttAllEntriesState(RttAt(rtt), rtte_pre.state)</code>
rtte_c_addr	pre: <code>(rtte_pre.state != UNASSIGNED</code> <code>&& rtte_pre.state != UNASSIGNED_NS)</code> post: <code>RttAllEntriesContiguous(RttAt(rtt), rtte_pre.addr, level)</code>

B4.3.38.4 Footprint

ID	Value
rtt_state	<code>GranuleAt(rtt).state</code>
rtte	<code>RttEntryAt(RttAt(walk.rtt_addr), entry_idx)</code>

DRAFT

B4.3.39 RMI_RTT_DESTROY command

Destroys a primary RTT.

See also:

- [A5.6 Realm Translation Table](#)
- [A5.6.9 RTT destruction](#)
- [B4.3.38 RMI_RTT_CREATE command](#)
- [B4.3.41 RMI_RTT_FOLD command](#)

B4.3.39.1 Interface

B4.3.39.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400015E
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	Base of the IPA range described by the RTT
level	X3	63:0	Int64	RTT level

B4.3.39.1.2 Context

The RMI_RTT_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level - 1, RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
walk_top	Address	RttSkipNonLiveEntries (RttAt (walk.rtt_addr), walk.level, ipa)	false	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.39.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
rtt	X1	63:0	Address	PA of the RTT which was destroyed
top	X2	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

The `rtt` output value is valid only when the command result is `RMI_SUCCESS`.

The values of the `result` and `top` output values for different command outcomes are summarized in the following table.

Scenario	result	top	walk.rtte.state
Target RTT exists and is not live	<code>RMI_SUCCESS</code>	<code>> ipa</code>	Before execution: <code>TABLE</code> After execution: <code>UNASSIGNED</code> and <code>RIPAS</code> is <code>DESTROYED</code>
Missing RTT	<code>(RMI_ERROR_RTT, < level)</code>	<code>> ipa</code>	<code>UNASSIGNED</code> or <code>UNASSIGNED_NS</code>
Block mapping at lower level	<code>(RMI_ERROR_RTT, < level)</code>	<code>== ipa</code>	<code>ASSIGNED</code> or <code>ASSIGNED_NS</code>
Live RTT at target level	<code>(RMI_ERROR_RTT, level)</code>	<code>== ipa</code>	<code>TABLE</code>
RTT walk was not performed, due to any other command failure	Another error code	<code>0</code>	Unknown

See also:

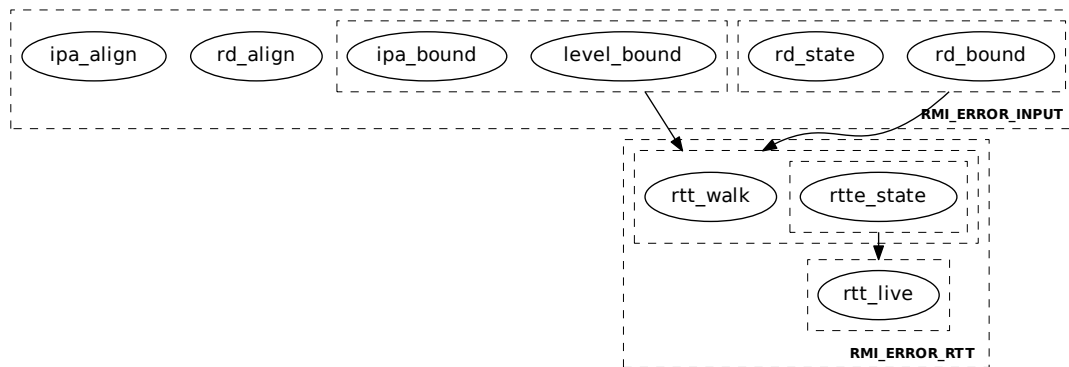
- [A5.6.8 RTTE liveness and RTT liveness](#)

B4.3.39.2 Failure conditions

ID	Condition
<code>rd_align</code>	pre: <code>!AddrIsGranuleAligned(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rd_bound</code>	pre: <code>!PaIsDelegable(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rd_state</code>	pre: <code>GranuleAt(rd).state != RD</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>level_bound</code>	pre: <code>(!RttLevelIsValid(realm, level) RttLevelIsStarting(realm, level))</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>ipa_align</code>	pre: <code>!AddrIsRttLevelAligned(ipa, level - 1)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>ipa_bound</code>	pre: <code>UInt(ipa) >= (2 ^ realm.ipa_width)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rtt_walk</code>	pre: <code>walk.level < level - 1</code> post: <code>(ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))</code>
<code>rtte_state</code>	pre: <code>walk.rtte.state != TABLE</code> post: <code>(ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))</code>
<code>rtt_live</code>	pre: <code>RttIsLive(RttAt(walk.rtte.addr))</code> post: <code>(ResultEqual(result, RMI_ERROR_RTT, level) && (top == ipa))</code>

B4.3.39.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[rtte_state] < [rtt_live]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.3.39.3 Success conditions

ID	Condition
state_prot	pre: <code>AddrIsProtected</code> (ipa, realm) post: walk.rtte.state == <code>UNASSIGNED</code>
ripas	pre: <code>AddrIsProtected</code> (ipa, realm) post: walk.rtte.ripas == <code>DESTROYED</code>
state_unprot	pre: <code>!AddrIsProtected</code> (ipa, realm) post: walk.rtte.state == <code>UNASSIGNED_NS</code>
rtt_state	<code>GranuleAt</code> (walk.rtte.addr).state == <code>DELEGATED</code>
rtt	rtt == walk.rtte.addr
top	top == walk_top

B4.3.39.4 Footprint

ID	Value
rtt_state	<code>GranuleAt</code> (walk.rtte.addr).state
rtte	<code>RttEntryAt</code> (<code>RttAt</code> (walk.rtt_addr), entry_idx)

B4.3.40 RMI_RTT_DEV_MEM_VALIDATE command

Completes a request made by the Realm to validate mappings to device memory from a target IPA range.

Issue In RMI_RTT_DEV_MEM_VALIDATE, consider how to combine:

- Modification of a range of RTT entries in a single command, and
- Checking of output address and HIPAS values against rec.ripas_dev_pa.

See also:

- [A5.5 Device memory mapping validation](#)
- [A9.5.3 Realm validation of device memory mappings](#)

B4.3.40.1 Interface

B4.3.40.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000163
rd	X1	63:0	Address	PA of the RD for the target Realm
rec_ptr	X2	63:0	Address	PA of the target REC
base	X3	63:0	Address	Base of target IPA region
top	X4	63:0	Address	Top of target IPA region

B4.3.40.1.2 Context

The RMI_RTT_DEV_MEM_VALIDATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
rec	RmmRec	RecAt (rec_ptr)	false	REC
pa_pre	Address	rec.dev_mem_pa	true	Output base address
walk	RmmRttWalkResult	RttWalk (realm, base, RMM_RTT_PAGE_LEVEL, RMM_RTT_TREE_PRIMARY)	false	RTT walk result
walk_top_pre	Address	RttSkipEntriesWithRipas (RttAt (walk.rtt_addr), walk.level, base, top, FALSE)	true	Top IPA of entries which have associated RIPAS values, starting from entry at which the RTT walk terminated

B4.3.40.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
out_top	X1	63:0	Address	Top IPA of range whose RIPAS was modified

The `out_top` output value is valid only when the command result is `RMI_SUCCESS`.

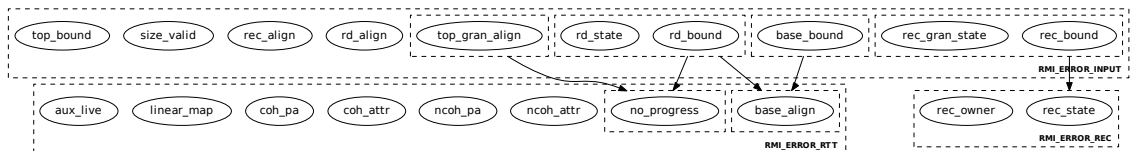
B4.3.40.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)
rec_align	pre: !AddrIsGranuleAligned (rec_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
rec_bound	pre: !PaIsDelegable (rec_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
rec_gran_state	pre: GranuleAt (rec_ptr).state != REC post: ResultEqual (result, RMI_ERROR_INPUT)
rec_state	pre: rec.state == REC_RUNNING post: ResultEqual (result, RMI_ERROR_REC)
rec_owner	pre: rec.owner != rd post: ResultEqual (result, RMI_ERROR_REC)
size_valid	pre: UInt (top) <= UInt (base) post: ResultEqual (result, RMI_ERROR_INPUT)
base_bound	pre: base != rec.dev_mem_addr post: ResultEqual (result, RMI_ERROR_INPUT)
top_bound	pre: UInt (top) > UInt (rec.dev_mem_top) post: ResultEqual (result, RMI_ERROR_INPUT)
base_align	pre: !AddrIsRttLevelAligned (base, walk.level) post: ResultEqual (result, RMI_ERROR_RTT , walk.level)
top_gran_align	pre: !AddrIsGranuleAligned (top) post: ResultEqual (result, RMI_ERROR_INPUT)
no_progress	pre: UInt (base) == UInt (walk_top_pre) post: ResultEqual (result, RMI_ERROR_RTT , walk.level)
ncoh_attr	pre: (rec.dev_mem_flags.coh == DEV_MEM_NON_COHERENT && !RttEntriesInRangeMemAttr (RttAt (walk.rtt_addr), walk.level, base, walk_top_pre, MEMATTR_NON_CACHEABLE)) post: ResultEqual (result, RMI_ERROR_RTT , walk.level)

ID	Condition
ncoh_pa	<pre>pre: (rec.dev_mem_flags.coh == DEV_MEM_NON_COHERENT && !RttEntriesInRangeNonCohDevMem(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre)) post: ResultEqual(result, RMI_ERROR_RTT, walk.level)</pre>
coh_attr	<pre>pre: (rec.dev_mem_flags.coh == DEV_MEM_COHERENT && !RttEntriesInRangeMemAttr(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre, MEMATTR_PASSTHROUGH)) post: ResultEqual(result, RMI_ERROR_RTT, walk.level)</pre>
coh_pa	<pre>pre: (rec.dev_mem_flags.coh == DEV_MEM_COHERENT && !RttEntriesInRangeCohDevMem(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre)) post: ResultEqual(result, RMI_ERROR_RTT, walk.level)</pre>
linear_map	<pre>pre: !RttEntriesInRangeOutputContiguous(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre, rec.dev_mem_pa) post: ResultEqual(result, RMI_ERROR_RTT, walk.level)</pre>
aux_live	<pre>pre: AddrRangeIsAuxLive(base, top, realm_pre) post: ResultEqual(result, RMI_ERROR_RTT, walk.level)</pre>

B4.3.40.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [base_align]
[rd_bound, rd_state] < [no_progress]
[rec_bound, rec_gran_state] < [rec_state, rec_owner]
[base_bound] < [base_align]
[top_gran_align] < [no_progress]
```



B4.3.40.3 Success conditions

ID	Condition
rtte_ripas	<code>RttEntriesInRangeRipas(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre, DEV)</code>
dev_mem_addr	<code>rec.dev_mem_addr == MinAddress(top, walk_top_pre)</code>
dev_mem_pa	<code>rec.dev_mem_pa == ToAddress(UInt(pa_pre) + (UInt(walk_top_pre) - UInt(base)))</code>
out_top	<code>out_top == MinAddress(top, walk_top_pre)</code>

B4.3.40.4 Footprint

ID	Value
rtte	<code>RttAt(walk.rtt_addr)</code>
dev_mem_addr	<code>rec.dev_mem_addr</code>

B4.3.41 RMI_RTT_FOLD command

Destroys a homogeneous primary RTT.

See also:

- [A5.6 Realm Translation Table](#)
- [A5.6.6 RTT folding](#)
- [B4.3.38 RMI_RTT_CREATE command](#)
- [B4.3.39 RMI_RTT_DESTROY command](#)

B4.3.41.1 Interface

B4.3.41.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000166
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	Base of the IPA range described by the RTT
level	X3	63:0	Int64	RTT level

B4.3.41.1.2 Context

The RMI_RTT_FOLD command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level - 1, RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
fold_pre	RmmRttEntry	RttFold (RttAt (walk.rtte.addr))	true	Result of folding RTT

B4.3.41.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
rtt	X1	63:0	Address	PA of the RTT which was destroyed

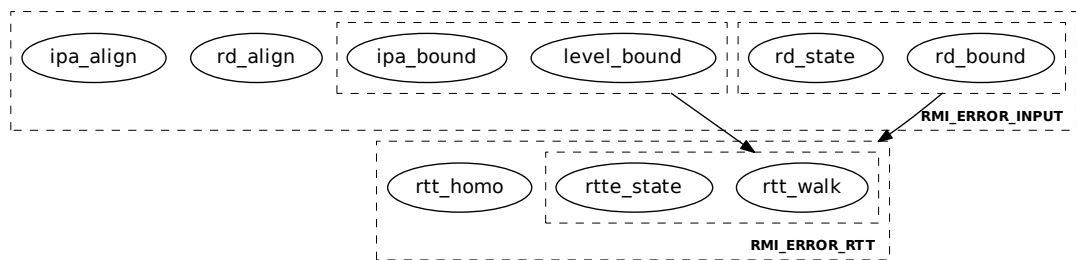
The `rtt` output value is valid only when the command result is `RMI_SUCCESS`.

B4.3.41.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
level_bound	pre: (!RttLevelIsValid(realm, level) RttLevelIsStarting(realm, level)) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsRttLevelAligned(ipa, level - 1) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: UInt(ipa) >= (2 ^ realm.ipa_width) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < level - 1 post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_state	pre: walk.rtte.state != TABLE post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtt_homo	pre: !RttIsHomogeneous(RttAt(walk.rtte.addr)) post: ResultEqual(result, RMI_ERROR_RTT, level)

B4.3.41.2.1 Failure condition ordering

[rd_bound, rd_state] < [rtt_walk, rtte_state, rtt_homo]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]



B4.3.41.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == fold_pre.state
rtte_addr	pre: (fold_pre.state != UNASSIGNED && fold_pre.state != UNASSIGNED_NS) post: walk.rtte.addr == fold_pre.addr

ID	Condition
rtte_attr_prot	pre: fold_pre.state == ASSIGNED post: (RttMemAttrEqual(walk.rtte, fold_pre, RTT_PROTECTED) && RttS2APEqual(walk.rtte, fold_pre, S2AP_INDIRECT))
rtte_attr_unprot	pre: fold_pre.state == ASSIGNED_NS post: (RttMemAttrEqual(walk.rtte, fold_pre, RTT_UNPROTECTED) && RttS2APEqual(walk.rtte, fold_pre, realm.rtt_s2ap_encoding))
rtte_ripas	pre: AddrIsProtected(ipa, realm) post: walk.rtte.ripas == fold_pre.ripas
rtt_state	GranuleAt(walk.rtte.addr).state == DELEGATED
rtt	rtt == walk.rtte.addr

B4.3.41.4 Footprint

ID	Value
rtt_state	GranuleAt(walk.rtte.addr).state
rtte	RttEntryAt(RttAt(walk.rtt_addr), entry_idx)

B4.3.42 RMI_RTT_INIT_RIPAS command

Set the RIPAS of a target IPA range to RAM, for a Realm in the REALM_NEW state.

See also:

- [A5.2.2 Realm IPA state](#)
- [D1.2.3 Initialize memory of New Realm flow](#)

B4.3.42.1 Interface

B4.3.42.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000168
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of target IPA region
top	X3	63:0	Address	Top of target IPA region

B4.3.42.1.2 Context

The RMI_RTT_INIT_RIPAS command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, base, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result
walk_top	Address	RttSkipEntriesIfNotState (RttAt (walk.rtt_addr), walk.level, base, top, UNASSIGNED)	false	Top IPA of UNASSIGNED entries, starting from entry at which the RTT walk terminated

B4.3.42.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
out_top	X1	63:0	Address	Top IPA of range whose RIPAS was modified

The out_top output value is valid only when the command result is RMI_SUCCESS.

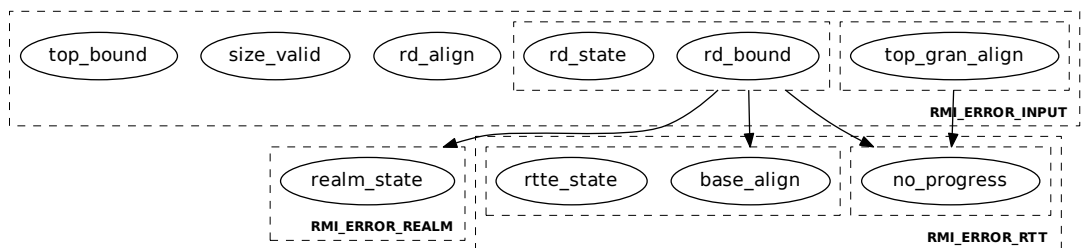
When the out_top output value is valid, it is aligned to the size of the address range described by the RTT entry at the level where the RTT walk terminated.

B4.3.42.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
size_valid	pre: UInt(top) <= UInt(base) post: ResultEqual(result, RMI_ERROR_INPUT)
top_bound	pre: !AddrIsProtected(ToAddress(UInt(top) - RMM_GRANULE_SIZE), realm_pre) post: ResultEqual(result, RMI_ERROR_INPUT)
realm_state	pre: realm_pre.state != REALM_NEW post: ResultEqual(result, RMI_ERROR_REALM)
base_align	pre: !AddrIsRttLevelAligned(base, walk.level) post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_state	pre: walk.rtte.state != UNASSIGNED post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
top_gran_align	pre: !AddrIsGranuleAligned(top) post: ResultEqual(result, RMI_ERROR_INPUT)
no_progress	pre: UInt(base) == UInt(walk_top) post: ResultEqual(result, RMI_ERROR_RTT, walk.level)

B4.3.42.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[rd_bound, rd_state] < [base_align, rtte_state]
[rd_bound, rd_state] < [no_progress]
[top_gran_align] < [no_progress]
```



B4.3.42.3 Success conditions

ID	Condition
rtte_ripas	<code>RttEntriesInRangeRipas (</code> <code> RttAt (walk.rtt_addr),</code> <code> walk.level,</code> <code> base, walk_top,</code> <code> RAM)</code>
rim	<code>realm.measurements[0] == RimExtendRipas (</code> <code> realm_pre, base, walk_top, walk.level)</code>
out_top	<code>out_top == walk_top</code>

B4.3.42.4 RMI_RTT_INIT_RIPAS extension of RIM

On successful execution of RMI_RTT_INIT_RIPAS, the new RIM value of the target Realm is calculated by the RMM as follows:

1. Allocate an [RmmMeasurementDescriptorRipas](#) data structure.
2. For each RTT entry in the range [base, top) described by the RMI_RTT_INIT_RIPAS input values:
 - a. Populate the measurement descriptor:
 - Set the desc_type field to the descriptor type.
 - Set the len field to the descriptor length.
 - Set the base field to the IPA of the RTT entry.
 - Set the top field to `Min(ipa + size, top)`, where
 - ipa is the IPA of the RTT entry
 - size is the size in bytes of the IPA region described by the RTT entry
 - top is the input value provided to the command
 - b. Using the RHA of the target Realm, compute the hash of the measurement descriptor. Set the RIM of the target Realm to this value, zero filling upper bytes if the RHA output is smaller than the size of the RIM.

See also:

- [A7.1.1 Realm Initial Measurement](#)
- [B3.77 RimExtendRipas function](#)
- [C2.19 RmmMeasurementDescriptorRipas type](#)

B4.3.42.5 Footprint

ID	Value
rtte	<code>RttAt (walk.rtt_addr)</code>
rim	<code>realm.measurements[0]</code>

B4.3.43 RMI_RTT_MAP_UNPROTECTED command

Creates a mapping from an Unprotected IPA to a Non-secure PA in a primary RTT.

See also:

- [A5.6 Realm Translation Table](#)
- [B4.3.47 RMI_RTT_UNMAP_UNPROTECTED command](#)

B4.3.43.1 Interface

B4.3.43.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400015F
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	IPA at which the Granule will be mapped in the target Realm
level	X3	63:0	Int64	RTT level
desc	X4	63:0	Bits64	RTTE descriptor

The layout and encoding of fields in the `desc` input value match “Attribute fields in stage 2 VMSAv8-64 Block and Page descriptors” in [Arm Architecture Reference Manual for A-Profile architecture](#) [3].

See also:

- [Arm Architecture Reference Manual for A-Profile architecture](#) [3]
- [A5.6.11.3 Stage 2 Access Permissions for an Unprotected IPA](#)
- [A5.6.12.3 Memory attributes for ASSIGNED_NS mappings](#)
- [B3.121 RttDescriptorIsValidForUnprotected function](#)

B4.3.43.1.2 Context

The RMI_RTT_MAP_UNPROTECTED command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (<code>rd</code>)	false	Realm
walk	RmmRttWalkResult	RttWalk (<code>realm, ipa, level,</code> <code>RMM_RTT_TREE_PRIMARY</code>)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (<code>ipa, walk.level</code>)	false	RTTE index
rtte	RmmRttEntry	RttDescriptorDecode (<code>desc,</code> <code>realm.rtt_s2ap_encoding</code> <code>↪</code>)	false	RTT entry

B4.3.43.1.3 Output values

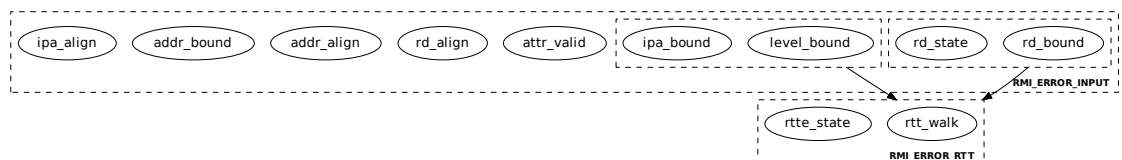
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.43.2 Failure conditions

ID	Condition
attr_valid	pre: !RttDescriptorIsValidForUnprotected (desc) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)
level_bound	pre: !RttLevelIsBlockOrPage (realm, level) post: ResultEqual (result, RMI_ERROR_INPUT)
addr_align	pre: !AddrIsRttLevelAligned (rtte.addr, level) post: ResultEqual (result, RMI_ERROR_INPUT)
addr_bound	pre: ((realm.feat_lpa2 == FEATURE_FALSE) && (UInt(rtte.addr) >= 2 ⁴⁸)) post: ResultEqual (result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsRttLevelAligned (ipa, level) post: ResultEqual (result, RMI_ERROR_INPUT)
ipa_bound	pre: (UInt(ipa) >= (2 ^ realm.ipa_width) AddrIsProtected (ipa, realm)) post: ResultEqual (result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < level post: ResultEqual (result, RMI_ERROR_RTT , walk.level)
rtte_state	pre: walk.rtte.state != UNASSIGNED_NS post: ResultEqual (result, RMI_ERROR_RTT , walk.level)

B4.3.43.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.3.43.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == ASSIGNED_NS
rtte_attr	walk.rtte.attr_unprot == rtte.attr_unprot
rtte_s2ap_direct	pre: realm.rtt_s2ap_encoding == S2AP_DIRECT post: (walk.rtte.s2ap_direct.read == rtte.s2ap_direct.read && walk.rtte.s2ap_direct.write == rtte.s2ap_direct.write)
rtte_s2ap_indirect	pre: realm.rtt_s2ap_encoding == S2AP_INDIRECT post: (walk.rtte.s2ap_indirect.base_index == rtte.s2ap_indirect.base_index && walk.rtte.s2ap_indirect.overlay_index == 15)
rtte_addr	walk.rtte.addr == rtte.addr

B4.3.43.4 Footprint

ID	Value
rtte	RttEntryAt(RttAt(walk.rtt_addr), entry_idx)

B4.3.44 RMI_RTT_READ_ENTRY command

Reads an entry from a primary RTT.

See also:

- [A5.6 Realm Translation Table](#)

B4.3.44.1 Interface

B4.3.44.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000161
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	Realm Address for which to read the RTTE
level	X3	63:0	Int64	RTT level at which to read the RTTE

B4.3.44.1.2 Context

The RMI_RTT_READ_ENTRY command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level, RMM_RTT_TREE_PRIMARY)	false	RTT walk result
rtte	RmmRttEntry	RttDescriptorDecode (desc, realm.rtt_s2ap_encoding ↪)	false	RTT entry value returned to Host

B4.3.44.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
walk_level	X1	63:0	UInt64	RTT level reached by the RTT walk
state	X2	7:0	RmiRttEntryState	State of RTTE reached by the walk
desc	X3	63:0	Bits64	RTTE descriptor
ripas	X4	7:0	RmiRipas	RIPAS of RTTE reached by the walk

The following unused bits of RMI_RTT_READ_ENTRY output values MBZ: X2[63:8], X4[63:8].

The layout and encoding of fields in the `rtte` output value match “Attribute fields in stage 2 VMSAv8-64 Block and Page descriptors” in [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#).

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)
- [A5.6.12 Memory attributes](#)

B4.3.44.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
level_bound	pre: !RttLevelIsValid(realm, level) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsRttLevelAligned(ipa, level) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: UInt(ipa) >= (2 ^ realm.ipa_width) post: ResultEqual(result, RMI_ERROR_INPUT)

B4.3.44.2.1 Failure condition ordering

The RMI_RTT_READ_ENTRY command does not have any failure condition orderings.

B4.3.44.3 Success conditions

ID	Condition
walk_level	walk_level == walk.level
state	state == RttEntryStateToRmi(walk.rtte.state)
state_invalid	pre: (walk.rtte.state == UNASSIGNED walk.rtte.state == UNASSIGNED_NS) post: (rtte.attr_unprot == Zeros(3) && rtte.s2ap_indirect.base_index == 0 && rtte.s2ap_indirect.overlay_index == 0 && rtte.s2ap_direct.read == RMM_FALSE && rtte.s2ap_direct.write == RMM_FALSE && rtte.addr == Zeros(ADDRESS_WIDTH))
state_prot	pre: (walk.rtte.state == ASSIGNED walk.rtte.state == ASSIGNED_DEV walk.rtte.state == ASSIGNED_VSMMU walk.rtte.state == TABLE) post: (rtte.attr_unprot == Zeros(3) && rtte.s2ap_indirect.base_index == 0 && rtte.s2ap_indirect.overlay_index == 0 && rtte.s2ap_direct.read == RMM_FALSE && rtte.s2ap_direct.write == RMM_FALSE && rtte.addr == walk.rtte.addr)

ID	Condition
state_unprot	<pre> pre: walk.rtte.state == ASSIGNED_NS post: (rtte.attr_unprot == walk.rtte.attr_unprot && rtte.s2ap_indirect.base_index == walk.rtte.s2ap_indirect.base_index && rtte.s2ap_indirect.overlay_index == 0 && rtte.s2ap_direct.read == walk.rtte.s2ap_direct.read && rtte.s2ap_direct.write == walk.rtte.s2ap_direct.write && rtte.addr == walk.rtte.addr) </pre>
state_io	<pre> pre: walk.rtte.state == ASSIGNED_DEV post: (rtte.attr_unprot == Zeros(3) && rtte.s2ap_indirect.base_index == 0 && rtte.s2ap_indirect.overlay_index == 0 && rtte.s2ap_direct.read == RMM_FALSE && rtte.s2ap_direct.write == RMM_FALSE && rtte.addr == walk.rtte.addr) </pre>
state_vsmmu	<pre> pre: walk.rtte.state == ASSIGNED_VSMMU post: (rtte.attr_unprot == Zeros(3) && rtte.s2ap_indirect.base_index == 0 && rtte.s2ap_indirect.overlay_index == 0 && rtte.s2ap_direct.read == RMM_FALSE && rtte.s2ap_direct.write == RMM_FALSE && rtte.addr == walk.rtte.addr) </pre>
ripas_prot	<pre> pre: (walk.rtte.state == UNASSIGNED walk.rtte.state == ASSIGNED) post: ripas == RipasToRmi(walk.rtte.ripas) </pre>
ripas_unprot	<pre> pre: (walk.rtte.state == UNASSIGNED_NS walk.rtte.state == ASSIGNED_NS) post: ripas == RMI_EMPTY </pre>

B4.3.44.4 Footprint

The RMI_RTT_READ_ENTRY command does not have any footprint.

B4.3.45 RMI_RTT_SET_RIPAS command

Completes a request made by the Realm to change the RIPAS of a target IPA range.

See also:

- [A5.4 RIPAS change](#)

B4.3.45.1 Interface

B4.3.45.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000169
rd	X1	63:0	Address	PA of the RD for the target Realm
rec_ptr	X2	63:0	Address	PA of the target REC
base	X3	63:0	Address	Base of target IPA region
top	X4	63:0	Address	Top of target IPA region

B4.3.45.1.2 Context

The RMI_RTT_SET_RIPAS command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
rec	RmmRec	RecAt (rec_ptr)	false	REC
walk	RmmRttWalkResult	RttWalk (realm, base, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result
ripas_pre	RmmRipas	walk.rtte.ripas	true	RIPAS before the command executed
walk_top_pre	Address	RttSkipEntriesWithRipas (RttAt (walk.rtt_addr), walk.level, base, top, rec.ripas_destroyed != CHANGE_DESTROYED)	true	Top IPA of entries which have associated RIPAS values, starting from entry at which the RTT walk terminated

B4.3.45.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
out_top	X1	63:0	Address	Top IPA of range whose RIPAS was modified

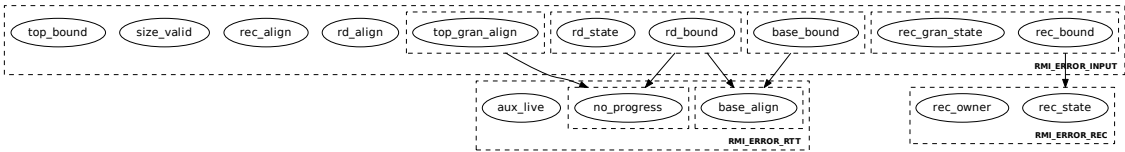
The `out_top` output value is valid only when the command result is `RMI_SUCCESS`.

B4.3.45.2 Failure conditions

ID	Condition
<code>rd_align</code>	pre: <code>!AddrIsGranuleAligned(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rd_bound</code>	pre: <code>!PaIsDelegable(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rd_state</code>	pre: <code>GranuleAt(rd).state != RD</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rec_align</code>	pre: <code>!AddrIsGranuleAligned(rec_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rec_bound</code>	pre: <code>!PaIsDelegable(rec_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rec_gran_state</code>	pre: <code>GranuleAt(rec_ptr).state != REC</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>rec_state</code>	pre: <code>rec.state == REC_RUNNING</code> post: <code>ResultEqual(result, RMI_ERROR_REC)</code>
<code>rec_owner</code>	pre: <code>rec.owner != rd</code> post: <code>ResultEqual(result, RMI_ERROR_REC)</code>
<code>size_valid</code>	pre: <code>UInt(top) <= UInt(base)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>base_bound</code>	pre: <code>base != rec.ripas_addr</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>top_bound</code>	pre: <code>UInt(top) > UInt(rec.ripas_top)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>base_align</code>	pre: <code>(!AddrIsRttLevelAligned(base, walk.level) && ripas_pre != rec.ripas_value)</code> post: <code>ResultEqual(result, RMI_ERROR_RTT, walk.level)</code>
<code>top_gran_align</code>	pre: <code>!AddrIsGranuleAligned(top)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
<code>no_progress</code>	pre: <code>(UInt(base) == UInt(walk_top_pre) && ripas_pre != rec.ripas_value)</code> post: <code>ResultEqual(result, RMI_ERROR_RTT, walk.level)</code>
<code>aux_live</code>	pre: <code>AddrRangeIsAuxLive(base, top, realm_pre)</code> post: <code>ResultEqual(result, RMI_ERROR_RTT, walk.level)</code>

B4.3.45.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [base_align]
[rd_bound, rd_state] < [no_progress]
[rec_bound, rec_gran_state] < [rec_state, rec_owner]
[base_bound] < [base_align]
[top_gran_align] < [no_progress]
```



B4.3.45.3 Success conditions

ID	Condition
rtte_ripas	<code>RttEntriesInRangeRipas (</code> <code>RttAt (walk.rtt_addr),</code> <code>walk.level,</code> <code>base, walk_top_pre,</code> <code>rec.ripas_value)</code>
ripas_addr	<code>rec.ripas_addr == MinAddress (top, walk_top_pre)</code>
out_top	<code>out_top == MinAddress (top, walk_top_pre)</code>

B4.3.45.4 Footprint

ID	Value
rtte	<code>RttAt (walk.rtt_addr)</code>
ripas_addr	<code>rec.ripas_addr</code>

B4.3.46 RMI_RTT_SET_S2AP command

Completes a request made by the Realm to change the S2AP of a target IPA range.

See also:

- [A10.3.3.2 Stage 2 Access Permissions change within a multi-Plane Realm](#)

B4.3.46.1 Interface

B4.3.46.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400018B
rd	X1	63:0	Address	PA of the RD for the target Realm
rec_ptr	X2	63:0	Address	PA of the target REC
base	X3	63:0	Address	Base of target IPA region
top	X4	63:0	Address	Top of target IPA region

B4.3.46.1.2 Context

The RMI_RTT_SET_S2AP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
rec	RmmRec	RecAt (rec_ptr)	false	REC
not_aligned	RmmRttWalkNotAligned	RttWalkAnyNotAligned (realm, base, top, RMM_RTT_PAGE_LEVEL)	false	RTT walk result which is not aligned to page level

B4.3.46.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
out_top	X1	63:0	Address	Top IPA of range whose S2AP was modified
rtt_tree	X2	63:0	UInt64	Index of RTT tree in which base alignment check failed

If `result` is `RMI_ERROR_RTT` or `RMI_ERROR_RTT_AUX` then the following are true:

- `out_top` is the IPA of the RTTE at which the base alignment check failed.
- `rtt_tree` is the index of the RTT in which the base alignment check failed.

B4.3.46.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
rec_align	pre: !AddrIsGranuleAligned(rec_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
rec_bound	pre: !PaIsDelegable(rec_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
rec_gran_state	pre: GranuleAt(rec_ptr).state != REC post: ResultEqual(result, RMI_ERROR_INPUT)
rec_state	pre: rec.state == REC_RUNNING post: ResultEqual(result, RMI_ERROR_REC)
rec_owner	pre: rec.owner != rd post: ResultEqual(result, RMI_ERROR_REC)
size_valid	pre: UInt(top) <= UInt(base) post: ResultEqual(result, RMI_ERROR_INPUT)
base_bound	pre: base != rec.s2ap_addr post: ResultEqual(result, RMI_ERROR_INPUT)
top_bound	pre: UInt(top) > UInt(rec.s2ap_top) post: ResultEqual(result, RMI_ERROR_INPUT)
top_gran_align	pre: !AddrIsGranuleAligned(top) post: ResultEqual(result, RMI_ERROR_INPUT)
base_align_pri	pre: (not_aligned.valid == RMM_TRUE && !AddrRangeIsWithin(base, top, AlignDownToRttLevel(not_aligned.addr, not_aligned.walk.level), AlignUpToRttLevel(not_aligned.addr, not_aligned.walk.level)) && not_aligned.index == RMM_RTT_TREE_PRIMARY && not_aligned.walk.rtte.s2ap_indirect.overlay_index != rec.s2ap_overlay_index) post: ResultEqual(result, RMI_ERROR_RTT, not_aligned.walk.level)

ID	Condition
base_align_aux	<pre> pre: (not_aligned.valid == RMM_TRUE && !AddrRangeIsWithin(base, top, AlignDownToRttLevel(not_aligned.addr, not_aligned.walk.level), AlignUpToRttLevel(not_aligned.addr, not_aligned.walk.level)) && not_aligned.index != RMM_RTT_TREE_PRIMARY && not_aligned.walk.rtte.s2ap_indirect.overlay_index != rec.s2ap_overlay_index) post: ResultEqual(result, RMI_ERROR_RTT_AUX, not_aligned.walk.level) </pre>

B4.3.46.2.1 Failure condition ordering

The RMI_RTT_SET_S2AP command does not have any failure condition orderings.

B4.3.46.3 Success conditions

ID	Condition
s2ap_addr	rec.s2ap_addr == out_top

B4.3.46.4 Footprint

The RMI_RTT_SET_S2AP command does not have any footprint.

B4.3.47 RMI_RTT_UNMAP_UNPROTECTED command

Removes a mapping at an Unprotected IPA.

See also:

- [A5.6 Realm Translation Table](#)
- [B4.3.43 RMI_RTT_MAP_UNPROTECTED command](#)

B4.3.47.1 Interface

B4.3.47.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000162
rd	X1	63:0	Address	PA of the RD for the target Realm
ipa	X2	63:0	Address	IPA at which the Granule is mapped in the target Realm
level	X3	63:0	Int64	RTT level

B4.3.47.1.2 Context

The RMI_RTT_UNMAP_UNPROTECTED command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, level, RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
walk_top	Address	RttSkipNonLiveEntries (RttAt (walk.rtt_addr), walk.level, ipa)	false	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.47.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
top	X1	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

The values of the `result` and `top` output values for different command outcomes are summarized in the following table.

Scenario	result	top	walk.rtte.state
ipa is mapped at the target level	RMI_SUCCESS	> ipa	Before execution: ASSIGNED_NS After execution: UNASSIGNED_NS
ipa is not mapped	(RMI_ERROR_RTT, <= level)	> ipa	UNASSIGNED_NS
ipa is mapped at a lower level	(RMI_ERROR_RTT, < level)	== ipa	ASSIGNED_NS
RTT walk was not performed, due to any other command failure	Another error code	0	Unknown

See also:

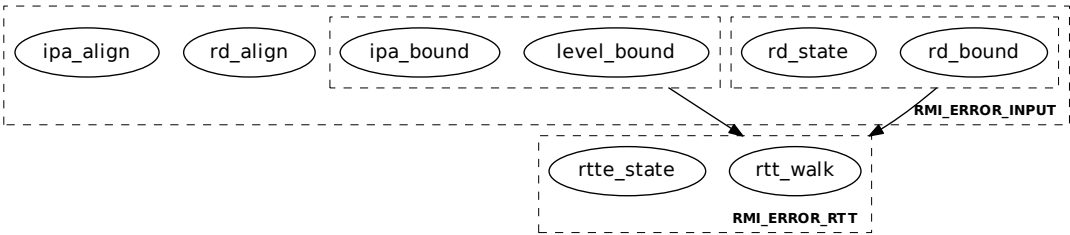
- [A5.6.8 RTTE liveness and RTT liveness](#)

B4.3.47.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsGranuleAligned(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
level_bound	pre: !RttLevelIsBlockOrPage(realm, level) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_align	pre: !AddrIsRttLevelAligned(ipa, level) post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: (UInt(ipa) >= (2 ^ realm.ipa_width) AddrIsProtected(ipa, realm)) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < level post: (ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))
rtte_state	pre: walk.rtte.state != ASSIGNED_NS post: (ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))

B4.3.47.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.3.47.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == UNASSIGNED_NS
top	top == walk_top

B4.3.47.4 Footprint

ID	Value
rtte	RttEntryAt(RttAt(walk.rtt_addr), entry_idx)

B4.3.48 RMI_VDEV_ABORT command

Abort device communication associated with a VDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.48.1 Interface

B4.3.48.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000185
vdev_ptr	X1	63:0	Address	PA of the VDEV

B4.3.48.1.2 Context

The RMI_VDEV_ABORT command operates on the following context.

Name	Type	Value	Before	Description
vdev	RmmVdev	VdevAt (vdev_ptr)	false	VDEV

B4.3.48.1.3 Output values

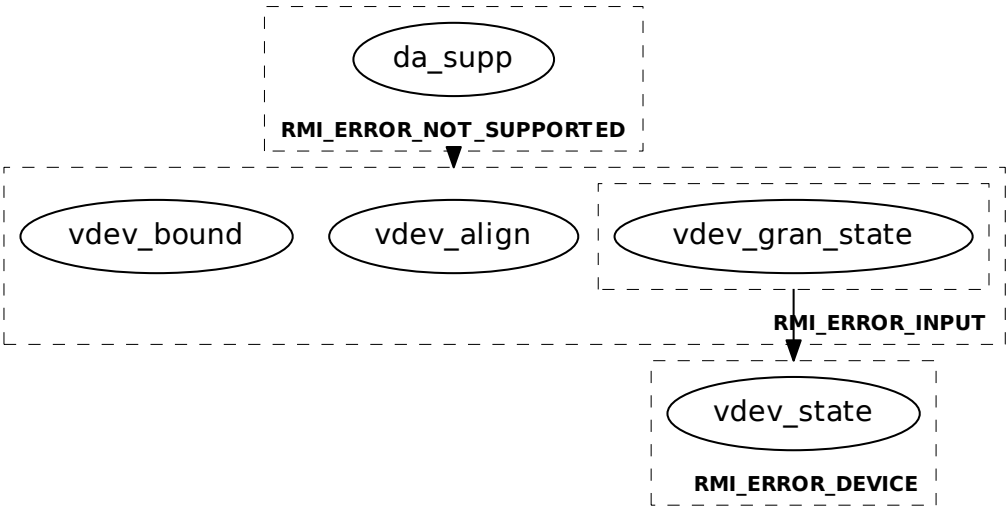
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.48.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
vdev_align	pre: !AddrIsGranuleAligned (vdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_bound	pre: !PaIsDelegable (vdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_gran_state	pre: GranuleAt (vdev_ptr) .state != VDEV post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_state	pre: vdev.state != VDEV_COMMUNICATING post: ResultEqual (result, RMI_ERROR_DEVICE)

B4.3.48.2.1 Failure condition ordering

[da_supp] < [vdev_align, vdev_bound, vdev_gran_state]
[vdev_gran_state] < [vdev_state]



B4.3.48.3 Success conditions

ID	Condition
state	vdev.state == VDEV_READY
comm_state	vdev.comm_state == DEV_COMM_IDLE

B4.3.48.4 Footprint

ID	Value
state	vdev.state
comm_state	vdev.comm_state

B4.3.49 RMI_VDEV_AUX_COUNT command

Get number of auxiliary Granules required for a VDEV.

B4.3.49.1 Interface

B4.3.49.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000160
pdev_flags	X1	63:0	Bits64	PDEV flags
vdev_flags	X2	63:0	Bits64	VDEV flags

B4.3.49.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
aux_count	X1	63:0	UInt64	Number of auxiliary Granules required for a VDEV

B4.3.49.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures().feat_da != FEATURE_TRUE post: ResultEqual(result, RMI_ERROR_NOT_SUPPORTED)

B4.3.49.3 Success conditions

ID	Condition
aux_count	aux_count == VdevAuxCount(RmiPdevFlagsDecode(pdev_flags), RmiVdevFlagsDecode(vdev_flags))

B4.3.49.4 Footprint

The RMI_VDEV_AUX_COUNT command does not have any footprint.

B4.3.50 RMI_VDEV_COMMUNICATE command

Perform device communication associated with a VDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.50.1 Interface

B4.3.50.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000186
pdev_ptr	X1	63:0	Address	PA of the PDEV
vdev_ptr	X2	63:0	Address	PA of the VDEV
data_ptr	X3	63:0	Address	PA of the communication data structure

U₀₄₀₃

An implementation may store state in a PDEV object which is required for communication with the child VDEVs. For this reason, and to simplify locking of PDEV and VDEV objects which may be performed by the implementation, a PDEV pointer is passed to RMI_VDEV_COMMUNICATE.

B4.3.50.1.2 Context

The RMI_VDEV_COMMUNICATE command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV
vdev	RmmVdev	VdevAt (vdev_ptr)	false	VDEV
data	RmiDevCommData	RmiDevCommDataAt (data_ptr)	false	Device communication object

B4.3.50.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

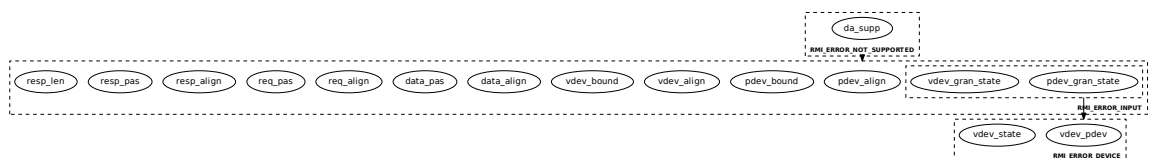
B4.3.50.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures ().feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
pdev_align	pre: !AddrIsGranuleAligned (pdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
pdev_bound	pre: !PaIsDelegable(pdev_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != PDEV post: ResultEqual(result, RMI_ERROR_INPUT)
vdev_align	pre: !AddrIsGranuleAligned(vdev_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
vdev_bound	pre: !PaIsDelegable(vdev_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != VDEV post: ResultEqual(result, RMI_ERROR_INPUT)
data_align	pre: !AddrIsGranuleAligned(data_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
data_pas	pre: !GranuleAccessPermitted(data_ptr, PAS_NS) post: ResultEqual(result, RMI_ERROR_INPUT)
req_align	pre: !AddrIsGranuleAligned(data.enter.req_addr) post: ResultEqual(result, RMI_ERROR_INPUT)
req_pas	pre: !GranuleAccessPermitted(data.enter.req_addr, PAS_NS) post: ResultEqual(result, RMI_ERROR_INPUT)
resp_align	pre: !AddrIsGranuleAligned(data.enter.resp_addr) post: ResultEqual(result, RMI_ERROR_INPUT)
resp_pas	pre: !GranuleAccessPermitted(data.enter.resp_addr, PAS_NS) post: ResultEqual(result, RMI_ERROR_INPUT)
resp_len	pre: data.enter.resp_len > RMM_GRANULE_SIZE post: ResultEqual(result, RMI_ERROR_INPUT)
vdev_pdev	pre: vdev.pdev != pdev_ptr post: ResultEqual(result, RMI_ERROR_DEVICE)
vdev_state	pre: (vdev.state != VDEV_COMMUNICATING && vdev.state != VDEV_STOPPING) post: ResultEqual(result, RMI_ERROR_DEVICE)

B4.3.50.2.1 Failure condition ordering

```
[da_supp] < [pdev_align, pdev_bound, pdev_gran_state, vdev_align,
             vdev_bound, vdev_gran_state, data_align, data_pas, req_align,
             req_pas, resp_align, resp_pas, resp_len]
[pdev_gran_state, vdev_gran_state] < [vdev_pdev, vdev_state]
```



B4.3.50.3 Success conditions

ID	Condition
comm_state	<code>vdev.comm_state == DeviceCommunicate(vdev, data)</code>
error	<pre>pre: (DeviceCommunicate(vdev, data) == DEV_COMM_ERROR && vdev.state == VDEV_COMMUNICATING) post: vdev.state == VDEV_ERROR</pre>
ready	<pre>pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE && vdev.state == VDEV_COMMUNICATING) post: vdev.state == VDEV_READY</pre>
stopped	<pre>pre: (DeviceCommunicate(vdev, data) != DEV_COMM_ACTIVE && vdev.state == VDEV_STOPPING) post: vdev.state == VDEV_STOPPED</pre>

B4.3.50.4 Footprint

ID	Value
state	<code>vdev.state</code>
comm_state	<code>vdev.comm_state</code>

B4.3.51 RMI_VDEV_COMPLETE command

Completes a pending VDEV request.

See also:

- [A4.3.12 REC exit due to VDEV request](#)
- [A9.2.2 Mapping from virtual device ID to VDEV object](#)

B4.3.51.1 Interface

B4.3.51.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400018E
rec_ptr	X1	63:0	Address	PA of the REC
vdev_ptr	X2	63:0	Address	PA of the VDEV

B4.3.51.1.2 Context

The RMI_VDEV_COMPLETE command operates on the following context.

Name	Type	Value	Before	Description
rec	RmmRec	RecAt (rec_ptr)	false	REC
vdev	RmmVdev	VdevAt (vdev_ptr)	false	VDEV

B4.3.51.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.51.2 Failure conditions

ID	Condition
rec_align	pre: !AddrIsGranuleAligned (rec_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
rec_bound	pre: !PaIsDelegable (rec_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
recv_state	pre: GranuleAt (rec_ptr).state != REC post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_align	pre: !AddrIsGranuleAligned (vdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_bound	pre: !PaIsDelegable (vdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
vdev_state	pre: <code>GranuleAt(vdev_ptr).state != VDEV</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pending	pre: <code>rec.pending != REC_PENDING_VDEV_REQUEST</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
owner	pre: <code>rec.owner != vdev.realm</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vdev_id	pre: <code>rec.vdev_id != vdev.vdev_id</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
inst_id	pre: <code>(rec.inst_id_valid == RMM_TRUE && rec.inst_id != vdev.inst_id)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>

B4.3.51.2.1 Failure condition ordering

The RMI_VDEV_COMPLETE command does not have any failure condition orderings.

B4.3.51.3 Success conditions

ID	Condition
pending	<code>rec.pending == REC_PENDING_NONE</code>

B4.3.51.4 Footprint

ID	Value
pend	<code>rec.pending</code>

B4.3.52 RMI_VDEV_CREATE command

Create a VDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.52.1 Interface

B4.3.52.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000187
rd	X1	63:0	Address	PA of the RD
pdev_ptr	X2	63:0	Address	PA of the PDEV
vdev_ptr	X3	63:0	Address	PA of the VDEV
params_ptr	X4	63:0	Address	PA of VDEV parameters

B4.3.52.1.2 Context

The RMI_VDEV_CREATE command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
realm	RmmRealm	RealmAt (rd)	false	Realm
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV
num_vdevs_pre	UInt64	pdev.num_vdevs	true	Number of VDEVs associated with the PDEV
vdev	RmmVdev	VdevAt (vdev_ptr)	false	VDEV
params	RmiVdevParams	RmiVdevParamsAt (params_ptr)	false	VDEV parameters
num_aux	UInt64	VdevAuxCount (PdevFlags (pdev) , params.flags)	false	Number of auxiliary Granules
rdev	RmmRdev	RdevFromInstId (realm, realm_pre.vdev_count)	false	RDEV

B4.3.52.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.52.2 Failure conditions

ID	Condition
da_supp	pre: <code>ImplFeatures().feat_da != FEATURE_TRUE</code> post: <code>ResultEqual(result, RMI_ERROR_NOT_SUPPORTED)</code>
rd_align	pre: <code>!AddrIsGranuleAligned(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_bound	pre: <code>!PaIsDelegable(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_state	pre: <code>GranuleAt(rd).state != RD</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pdev_align	pre: <code>!AddrIsGranuleAligned(pdev_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pdev_bound	pre: <code>!PaIsDelegable(pdev_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pdev_gran_state	pre: <code>GranuleAt(pdev_ptr).state != PDEV</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pdev_state	pre: <code>pdev.state != PDEV_READY</code> post: <code>ResultEqual(result, RMI_ERROR_DEVICE)</code>
vdev_align	pre: <code>!AddrIsGranuleAligned(vdev_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vdev_bound	pre: <code>!PaIsDelegableDram(vdev_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vdev_gran_state	pre: <code>GranuleAt(vdev_ptr).state != DELEGATED</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
params_align	pre: <code>!AddrIsGranuleAligned(params_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
params_pas	pre: <code>!GranuleAccessPermitted(params_ptr, PAS_NS)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
params_valid	pre: <code>!RmiVdevParamsIsValid(params_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
da_en	pre: <code>realm.feat_da != FEATURE_TRUE</code> post: <code>ResultEqual(result, RMI_ERROR_REALM)</code>
num_aux	pre: <code>params.num_aux != num_aux</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
aux_align	pre: <code>!AuxAligned32(params.aux, params.num_aux)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
aux_alias	pre: <code>AuxAlias32(vdev_ptr, params.aux, params.num_aux)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
aux_state	pre: <code>!AuxStateEqual32(params.aux, params.num_aux, DELEGATED)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
tdi_id_free	pre: <code>!TdiIdIsFree(params.tdi_id, pdev.segment_id)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
tdi_id_bound	pre: <code>(UInt(params.tdi_id) < UInt(pdev.rid_base) UInt(params.tdi_id) >= UInt(pdev.rid_top))</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>

ID	Condition
vsmmu_align	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && !AddrIsGranuleAligned(params.vsmmu_addr)) post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_bound	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && !PaIsDelegable(params.vsmmu_addr)) post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_state	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && GranuleAt(params.vsmmu_addr).state != VSMMU) post: ResultEqual(result, RMI_ERROR_INPUT)
vsid_free	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && !VsidIsFree(VsmmuAt(params.vsmmu_addr), params.vsid)) post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_compat	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && !PdevVsmmuIsCompatible(pdev, VsmmuAt(params.vsmmu_addr))) post: ResultEqual(result, RMI_ERROR_INPUT)

B4.3.52.2.1 Failure condition ordering

```
[da_supp] < [rd_align, rd_bound, pdev_bound, pdev_gran_state,
vdev_align, vdev_bound, vdev_gran_state, params_align,
params_pas, params_valid, num_aux, aux_align, aux_alias,
aux_state, vsmmu_align, vsmmu_bound, vsmmu_state, vsid_free]
[da_supp] < [pdev_gran_state]
[da_supp] < [rd_state]
[pdev_gran_state, vsmmu_state] < [vsmmu_compat]
[pdev_gran_state] < [pdev_state]
[rd_state] < [da_en]
```



B4.3.52.3 Success conditions

ID	Condition
pdev_num_vdevs	pdev.num_vdevs == num_vdevs_pre + 1
gran_state	GranuleAt(vdev_ptr).state == VDEV
vdev_id	vdev.vdev_id == params.vdev_id
tdi_id	vdev.tdi_id == params.tdi_id
pdev	vdev.pdev == pdev_ptr
realm	vdev.realm == rd
state	vdev.state == VDEV_READY

ID	Condition
comm_state	vdev.comm_state == DEV_COMM_IDLE
rdev_state	rdev.state == RDEV_UNLOCKED
rdev_op	rdev.operation == RDEV_OP_NONE
rdev_vdev_ptr	rdev.vdev_ptr == vdev_ptr
aux	AuxEqual32(vdev.aux, params.aux, num_aux)
num_aux	vdev.num_aux == num_aux
aux_state	AuxStateEqual32(vdev.aux, num_aux, VDEV_AUX)
tdi_id_used	!TdiIdIsFree(params.tdi_id, pdev.segment_id)
inst_id	vdev.inst_id == realm_pre.vdev_count
vsmmu	Equal(vdev.vsmmu, params.flags.VSMMU)
vsmmu_addr	pre: params.flags.VSMMU == RMI_FEATURE_TRUE post: vdev.vsmmu_addr == params.vsmmu_addr
vsid	pre: params.flags.VSMMU == RMI_FEATURE_TRUE post: vdev.vsid == params.vsid
vsid_alloc	pre: params.flags.VSMMU == RMI_FEATURE_TRUE post: !VsidIsFree(VsmmuAt(params.vsmmu_addr), params.vsid)
realm_num_vdevs	realm.num_vdevs == realm_pre.num_vdevs + 1
realm_vdev_count	realm.vdev_count == realm_pre.vdev_count + 1

B4.3.52.4 Footprint

ID	Value
state	GranuleAt(vdev_ptr).state
aux_state	AuxStates(vdev.aux, num_aux)
pdev_num_vdevs	pdev.num_vdevs
realm_num_vdevs	realm.num_vdevs
realm_vdev_count	realm.vdev_count

B4.3.53 RMI_VDEV_DESTROY command

Destroy a VDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.53.1 Interface

B4.3.53.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000188
rd	X1	63:0	Address	PA of the RD
pdev_ptr	X2	63:0	Address	PA of the PDEV
vdev_ptr	X3	63:0	Address	PA of the VDEV

B4.3.53.1.2 Context

The RMI_VDEV_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	RealmAt (rd)	true	Realm
realm	RmmRealm	RealmAt (rd)	false	Realm
vdev_pre	RmmVdev	VdevAt (vdev_ptr)	true	VDEV
pdev_pre	RmmPdev	PdevAt (pdev_ptr)	true	PDEV
pdev	RmmPdev	PdevAt (pdev_ptr)	false	PDEV

B4.3.53.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

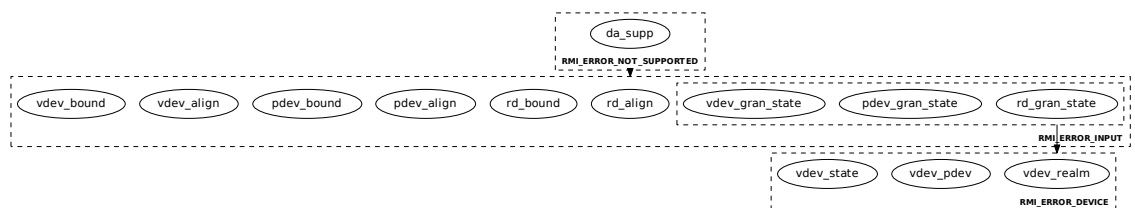
B4.3.53.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
rd_gran_state	pre: <code>GranuleAt(rd).state != RD</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pdev_align	pre: <code>!AddrIsGranuleAligned(pdev_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pdev_bound	pre: <code>!PaIsDelegable(pdev_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
pdev_gran_state	pre: <code>GranuleAt(pdev_ptr).state != PDEV</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vdev_align	pre: <code>!AddrIsGranuleAligned(vdev_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vdev_bound	pre: <code>!PaIsDelegable(vdev_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vdev_gran_state	pre: <code>GranuleAt(vdev_ptr).state != VDEV</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vdev_realm	pre: <code>vdev_pre.realm != rd</code> post: <code>ResultEqual(result, RMI_ERROR_DEVICE)</code>
vdev_pdev	pre: <code>vdev_pre.pdev != pdev_ptr</code> post: <code>ResultEqual(result, RMI_ERROR_DEVICE)</code>
vdev_state	pre: <code>vdev_pre.state != VDEV_STOPPED</code> post: <code>ResultEqual(result, RMI_ERROR_DEVICE)</code>

B4.3.53.2.1 Failure condition ordering

```
[da_supp] < [rd_align, rd_bound, rd_gran_state, pdev_align,
             pdev_bound, pdev_gran_state, vdev_align, vdev_bound,
             vdev_gran_state]
[rd_gran_state, pdev_gran_state, vdev_gran_state] < [vdev_realm,
             vdev_pdev, vdev_state]
```



B4.3.53.3 Success conditions

ID	Condition
gran_state	<code>GranuleAt(vdev_ptr).state == DELEGATED</code>
aux_state	<code>AuxStateEqual32(vdev_pre.aux, vdev_pre.num_aux, DELEGATED)</code>
tdi_id_free	<code>TdiIdIsFree(vdev_pre.tdi_id, pdev_pre.segment_id)</code>
realm_num_vdevs	<code>realm.num_vdevs == realm_pre.num_vdevs - 1</code>

ID	Condition
pdev_num_vdevs	<code>pdev.num_vdevs == pdev_pre.num_vdevs - 1</code>
vsid_free	<code>pre: vdev_pre.vsmmu == FEATURE_TRUE</code> <code>post: VsidIsFree(VsmmuAt(vdev_pre.vsmmu_addr), vdev_pre.vsid)</code>

B4.3.53.4 Footprint

ID	Value
state	<code>GranuleAt(vdev_ptr).state</code>
aux_state	<code>AuxStates(vdev_pre.aux, vdev_pre.num_aux)</code>
realm_num_vdevs	<code>realm.num_vdevs</code>
pdev_num_vdevs	<code>pdev.num_vdevs</code>

DRAFT

B4.3.54 RMI_VDEV_GET_STATE command

Get state of a VDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.54.1 Interface

B4.3.54.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000189
vdev_ptr	X1	63:0	Address	PA of the VDEV

B4.3.54.1.2 Context

The RMI_VDEV_GET_STATE command operates on the following context.

Name	Type	Value	Before	Description
vdev	RmmVdev	VdevAt (vdev_ptr)	false	VDEV

B4.3.54.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
state	X1	7:0	RmiVdevState	VDEV state

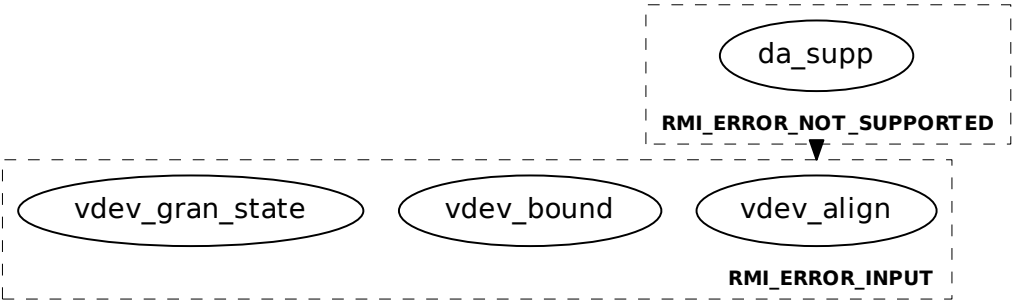
The following unused bits of RMI_VDEV_GET_STATE output values MBZ: X1[63:8].

B4.3.54.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
vdev_align	pre: !AddrIsGranuleAligned (vdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_bound	pre: !PaIsDelegable (vdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_gran_state	pre: GranuleAt (vdev_ptr) .state != VDEV post: ResultEqual (result, RMI_ERROR_INPUT)

B4.3.54.2.1 Failure condition ordering

[da_supp] < [vdev_align, vdev_bound, vdev_gran_state]



B4.3.54.3 Success conditions

ID	Condition
state	<code>Equal(state, vdev.state)</code>

B4.3.54.4 Footprint

The RMI_VDEV_GET_STATE command does not have any footprint.

B4.3.55 RMI_VDEV_STOP command

Stop a VDEV.

See also:

- [Chapter A9 Realm device assignment](#)

B4.3.55.1 Interface

B4.3.55.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400018A
vdev_ptr	X1	63:0	Address	PA of the VDEV

B4.3.55.1.2 Context

The RMI_VDEV_STOP command operates on the following context.

Name	Type	Value	Before	Description
vdev	RmmVdev	VdevAt (vdev_ptr)	false	VDEV

B4.3.55.1.3 Output values

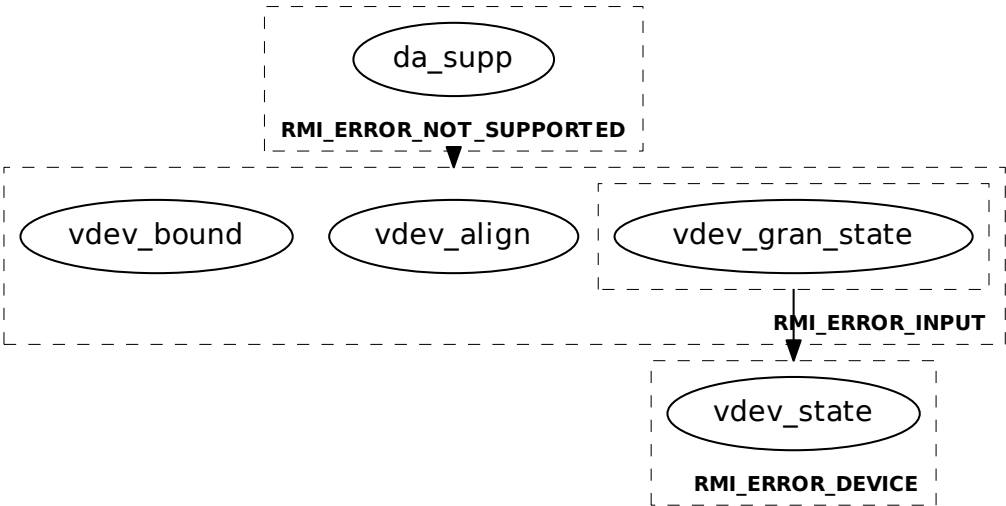
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

B4.3.55.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
vdev_align	pre: !AddrIsGranuleAligned (vdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_bound	pre: !PaIsDelegable (vdev_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_gran_state	pre: GranuleAt (vdev_ptr) .state != VDEV post: ResultEqual (result, RMI_ERROR_INPUT)
vdev_state	pre: (vdev.state != VDEV_READY && vdev.state != VDEV_ERROR) post: ResultEqual (result, RMI_ERROR_DEVICE)

B4.3.55.2.1 Failure condition ordering

[da_supp] < [vdev_align, vdev_bound, vdev_gran_state]
[vdev_gran_state] < [vdev_state]



B4.3.55.3 Success conditions

ID	Condition
state	vdev.state == VDEV_STOPPING
comm_state	vdev.comm_state == DEV_COMM_PENDING

B4.3.55.4 Footprint

ID	Value
state	vdev.state
comm_state	vdev.comm_state

B4.3.56 RMI_VERSION command

Allows the Host and the RMM to determine whether there exists a mutually acceptable revision of the RMM via which the two components can communicate.

On calling this command, the Host provides a requested RMI version.

The output values include a status code and two revisions which are supported by the RMM: a *lower revision* and a *higher revision*.

- The *higher revision* value is the highest interface revision which is supported by the RMM.
- The *lower revision* is less than or equal to the *higher revision*.

The status code and *lower revision* output values indicate which of the following is true, in order of precedence:

- a) The RMM supports an interface revision which is compatible with the requested revision.
 - The status code is RMI_SUCCESS.
 - The *lower revision* is equal to the requested revision.
- b) The RMM does not support an interface revision which is compatible with the requested revision. The RMM supports an interface revision which is incompatible with and less than the requested revision.
 - The status code is RMI_ERROR_INPUT.
 - The *lower revision* is the highest interface revision which is both less than the requested revision and supported by the RMM.
- c) The RMM does not support an interface revision which is compatible with the requested revision. The RMM supports an interface revision which is incompatible with and greater than the requested revision.
 - The status code is RMI_ERROR_INPUT.
 - The *lower revision* is equal to the *higher revision*.

See also:

- [Chapter B2 Interface versioning](#)
- [B4.1 RMI version](#)

B4.3.56.1 Interface

B4.3.56.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000150
req	X1	63:0	RmiInterfaceVersion	Requested interface revision

B4.3.56.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
lower	X1	63:0	RmiInterfaceVersion	Lower supported interface revision
higher	X2	63:0	RmiInterfaceVersion	Higher supported interface revision

B4.3.56.2 Failure conditions

ID	Condition
incompat_lower	<pre>pre: (!RmiVersionIsSupported(req) && RmiVersionLowerIsSupported(req)) post: (ResultEqual(result, RMI_ERROR_INPUT) && VersionEqual(lower, RmiVersionHighestBelow(req)) && VersionEqual(higher, RmiVersionHighest()))</pre>
incompat_higher	<pre>pre: (!RmiVersionIsSupported(req) && !RmiVersionLowerIsSupported(req) && RmiVersionHigherIsSupported(req)) post: (ResultEqual(result, RMI_ERROR_INPUT) && VersionEqual(lower, higher) && VersionEqual(higher, RmiVersionHighest()))</pre>

B4.3.56.2.1 Failure condition ordering

The RMI_VERSION command does not have any failure condition orderings.

B4.3.56.3 Success conditions

ID	Condition
lower	<code>VersionEqual(lower, req)</code>
higher	<code>VersionEqual(higher, RmiVersionHighest())</code>

B4.3.56.4 Footprint

The RMI_VERSION command does not have any footprint.

B4.3.57 RMI_VSMMU_CREATE command

Create a VSMMU.

See also:

- [A9.6 Virtual SMMU](#)
- [B4.3.58 RMI_VSMMU_DESTROY command](#)

B4.3.57.1 Interface

B4.3.57.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400016A
rd	X1	63:0	Address	PA of the RD
vsmmu_ptr	X2	63:0	Address	PA of the VSMMU
params_ptr	X3	63:0	Address	PA of VSMMU parameters

B4.3.57.1.2 Context

The RMI_VSMMU_CREATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
vsmmu	RmmVsmmu	VsmmuAt (vsmmu_ptr)	false	VSMMU
params	RmiVsmmuParams	RmiVsmmuParamsAt (params_ptr)	false	VSMMU parameters

B4.3.57.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

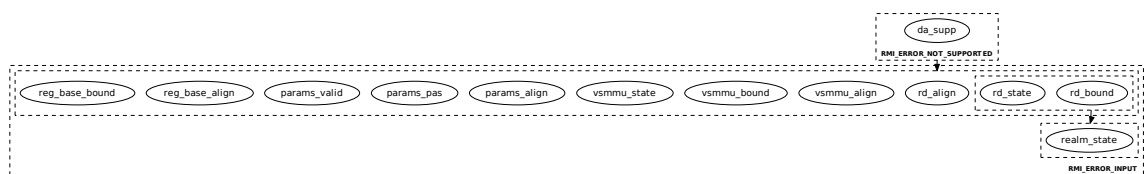
B4.3.57.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures ().feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
realm_state	pre: realm.state != REALM_NEW post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_align	pre: !AddrIsGranuleAligned(vsmmu_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_bound	pre: !PaIsDelegableDram(vsmmu_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_state	pre: GranuleAt(vsmmu_ptr).state != DELEGATED post: ResultEqual(result, RMI_ERROR_INPUT)
params_align	pre: !AddrIsGranuleAligned(params_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
params_pas	pre: !GranuleAccessPermitted(params_ptr, PAS_NS) post: ResultEqual(result, RMI_ERROR_INPUT)
params_valid	pre: !RmiVsmmuParamsIsValid(params_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
reg_base_align	pre: (!AddrIsGranuleAligned(params.reg_base) !AddrIsGranuleAligned(params.reg_top)) post: ResultEqual(result, RMI_ERROR_INPUT)
reg_base_bound	pre: (!AddrIsProtected(params.reg_base, realm) !AddrIsProtected(params.reg_top, realm) UInt(params.reg_top) <= UInt(params.reg_base)) post: ResultEqual(result, RMI_ERROR_INPUT)

B4.3.57.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[da_supp] < [rd_align, rd_bound, rd_state, vsmmu_align, vsmmu_bound,
vsmmu_state, params_align, params_pas, params_valid,
reg_base_align, reg_base_bound]
```



B4.3.57.3 Success conditions

ID	Condition
gran_state	GranuleAt(vsmmu_ptr).state == VSMMU
realm	vsmmu.realm == rd
reg_base	vsmmu.reg_base == params.reg_base
reg_top	vsmmu.reg_top == params.reg_top
aidr	vsmmu.aidr == params.aidr

ID	Condition
idr	<pre>(vsmmu.idr[0] == params.idr[0] && vsmmu.idr[1] == params.idr[1] && vsmmu.idr[2] == params.idr[2] && vsmmu.idr[3] == params.idr[3] && vsmmu.idr[4] == params.idr[4] && vsmmu.idr[5] == params.idr[5] && vsmmu.idr[6] == params.idr[6])</pre>

B4.3.57.4 Footprint

ID	Value
state	<code>GranuleAt(vsmmu_ptr).state</code>

DRAFT

B4.3.58 RMI_VSMMU_DESTROY command

Destroy a VSMMU.

See also:

- [A9.6 Virtual SMMU](#)
- [B4.3.57 RMI_VSMMU_CREATE command](#)

B4.3.58.1 Interface

B4.3.58.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400016B
rd	X1	63:0	Address	PA of the RD
vsmmu_ptr	X2	63:0	Address	IPA of the VSMMU

B4.3.58.1.2 Context

The RMI_VSMMU_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
vsmmu	RmmVsmmu	VsmmuAt (vsmmu_ptr)	false	VSMMU

B4.3.58.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

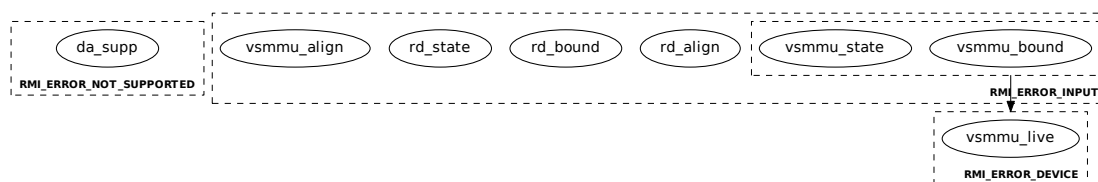
B4.3.58.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures ().feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_bound	pre: !PaIsDelegable (rd) post: ResultEqual (result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt (rd).state != RD post: ResultEqual (result, RMI_ERROR_INPUT)
vsmmu_align	pre: !AddrIsGranuleAligned (vsmmu_ptr) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
vsmmu_bound	pre: <code>!PaIsDelegable(vsmmu_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vsmmu_state	pre: <code>GranuleAt(vsmmu_ptr).state != VSMMU</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
vsmmu_live	pre: <code>VsmmuIsLive(vsmmu_ptr)</code> post: <code>ResultEqual(result, RMI_ERROR_DEVICE)</code>

B4.3.58.2.1 Failure condition ordering

`[vsmmu_bound, vsmmu_state] < [vsmmu_live]`



B4.3.58.3 Success conditions

ID	Condition
gran_state	<code>GranuleAt(vsmmu_ptr).state == DELEGATED</code>

B4.3.58.4 Footprint

ID	Value
state	<code>GranuleAt(vsmmu_ptr).state</code>

B4.3.59 RMI_VSMMU_MAP command

Create a VSMMU mapping.

See also:

- [A9.6 Virtual SMMU](#)
- [B4.3.60 RMI_VSMMU_UNMAP command](#)

B4.3.59.1 Interface

B4.3.59.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400016C
rd	X1	63:0	Address	PA of the RD
vsmmu_ptr	X2	63:0	Address	PA of the VSMMU
ipa	X3	63:0	Address	IPA at which to create the mapping

B4.3.59.1.2 Context

The RMI_VSMMU_MAP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
vsmmu	RmmVsmmu	VsmmuAt (vsmmu_ptr)	false	VSMMU
walk	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index

B4.3.59.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status

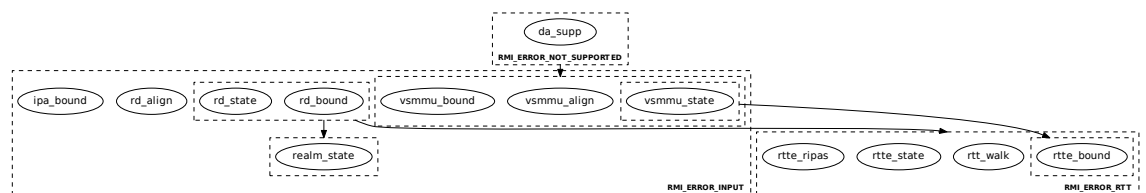
B4.3.59.2 Failure conditions

ID	Condition
da_supp	pre: ImplFeatures () .feat_da != FEATURE_TRUE post: ResultEqual (result, RMI_ERROR_NOT_SUPPORTED)
rd_align	pre: !AddrIsGranuleAligned (rd) post: ResultEqual (result, RMI_ERROR_INPUT)

ID	Condition
rd_bound	pre: !PaIsDelegable(rd) post: ResultEqual(result, RMI_ERROR_INPUT)
rd_state	pre: GranuleAt(rd).state != RD post: ResultEqual(result, RMI_ERROR_INPUT)
realm_state	pre: realm.state != REALM_NEW post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_align	pre: !AddrIsGranuleAligned(vsmmu_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_bound	pre: !PaIsDelegable(vsmmu_ptr) post: ResultEqual(result, RMI_ERROR_INPUT)
vsmmu_state	pre: GranuleAt(vsmmu_ptr).state != VSMMU post: ResultEqual(result, RMI_ERROR_INPUT)
ipa_bound	pre: UInt(ipa) < UInt(vsmmu.reg_base) post: ResultEqual(result, RMI_ERROR_INPUT)
rtt_walk	pre: walk.level < RMM_RTT_PAGE_LEVEL post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_state	pre: walk.rtte.state != UNASSIGNED post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_ripas	pre: walk.rtte.ripas != EMPTY post: ResultEqual(result, RMI_ERROR_RTT, walk.level)
rtte_bound	pre: (UInt(ipa) + (RttLevelSize(walk.level) - 1) >= UInt(vsmmu.reg_top)) post: ResultEqual(result, RMI_ERROR_RTT, walk.level)

B4.3.59.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[rd_bound, rd_state] < [rtt_walk, rtte_state, rtte_ripas, rtte_bound]
[vsmmu_state] < [rtte_bound]
[da_supp] < [vsmmu_align, vsmmu_bound, vsmmu_state]
```



B4.3.59.3 Success conditions

ID	Condition
rtte_state	walk.rtte.state == ASSIGNED_VSMMU
rtte_addr	walk.rtte.addr == vsmmu_ptr

B4.3.59.4 Footprint

ID	Value
rtte	<code>RttEntryAt(RttAt(walk.rtt_addr), entry_idx)</code>

DRAFT

B4.3.60 RMI_VSMMU_UNMAP command

Remove a VSMMU mapping.

See also:

- [A9.6 Virtual SMMU](#)
- [B4.3.59 RMI_VSMMU_MAP command](#)

B4.3.60.1 Interface

B4.3.60.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400016D
rd	X1	63:0	Address	PA of the RD
ipa	X2	63:0	Address	IPA at which to remove the mapping

B4.3.60.1.2 Context

The RMI_VSMMU_UNMAP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	RealmAt (rd)	false	Realm
walk	RmmRttWalkResult	RttWalk (realm, ipa, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result
entry_idx	UInt64	RttEntryIndex (ipa, walk.level)	false	RTTE index
walk_top	Address	RttSkipNonLiveEntries (RttAt (walk.rtt_addr), walk.level, ipa)	false	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.60.1.3 Output values

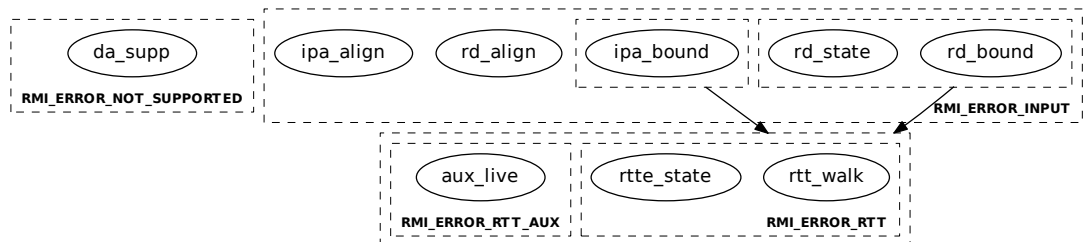
Name	Register	Bits	Type	Description
result	X0	63:0	RmiCommandReturnCode	Command return status
vsmmu	X1	63:0	Address	PA of the VSMMU Granule which was unmapped
top	X2	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.3.60.2 Failure conditions

ID	Condition
da_supp	pre: <code>ImplFeatures().feat_da != FEATURE_TRUE</code> post: <code>ResultEqual(result, RMI_ERROR_NOT_SUPPORTED)</code>
rd_align	pre: <code>!AddrIsGranuleAligned(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_bound	pre: <code>!PaIsDelegable(rd)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rd_state	pre: <code>GranuleAt(rd).state != RD</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
ipa_align	pre: <code>!AddrIsGranuleAligned(ipa)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
ipa_bound	pre: <code>!AddrIsProtected(ipa, realm)</code> post: <code>ResultEqual(result, RMI_ERROR_INPUT)</code>
rtt_walk	pre: <code>walk.level < RMM_RTT_PAGE_LEVEL</code> post: <code>(ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))</code>
rtte_state	pre: <code>walk.rtte.state != ASSIGNED_VSMMU</code> post: <code>(ResultEqual(result, RMI_ERROR_RTT, walk.level) && (top == walk_top))</code>
aux_live	pre: <code>AddrIsAuxLive(ipa, realm)</code> post: <code>ResultEqual(result, RMI_ERROR_RTT_AUX, 0)</code>

B4.3.60.2.1 Failure condition ordering

[rd_bound, rd_state] < [rtt_walk, rtte_state, aux_live]
[ipa_bound] < [rtt_walk, rtte_state, aux_live]



B4.3.60.3 Success conditions

ID	Condition
rtte_state	<code>walk.rtte.state == UNASSIGNED</code>
ripas_ram	pre: <code>walk.rtte.ripas == DEV</code> post: <code>walk.rtte.ripas == DESTROYED</code>
vsmmu	<code>vsmmu == walk.rtte.addr</code>
top	<code>top == walk_top</code>

B4.3.60.4 Footprint

ID	Value
data_state	<code>GranuleAt(walk.rtte.addr).state</code>
rtte	<code>RttEntryAt(RttAt(walk.rtt_addr), entry_idx)</code>

DRAFT

B4.4 RMI types

This section defines types which are used in the RMI interface.

B4.4.1 RmiAddressRange type

The RmiAddressRange structure contains address range.

The RmiAddressRange structure is a [concrete type](#).

The width of the RmiAddressRange structure is 16 (0x10) bytes.

The members of the RmiAddressRange structure are shown in the following table.

Name	Byte offset	Type	Description
base	0x0	Address	Base of address range (inclusive)
top	0x8	Address	Top of address range (exclusive)

The RmiAddressRange structure is used in the following types:

- [RmiPdevParams](#)

B4.4.2 RmiBoolean type

The RmiBoolean enumeration represents a boolean value.

The RmiBoolean enumeration is a [concrete type](#).

The width of the RmiBoolean enumeration is 1 bits.

The values of the RmiBoolean enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_FALSE	False
1	RMI_TRUE	True

The RmiBoolean enumeration is used in the following types:

- [RmiDevCommExitFlags](#)

B4.4.3 RmiCommandReturnCode type

The RmiCommandReturnCode fieldset contains a return code from an RMI command.

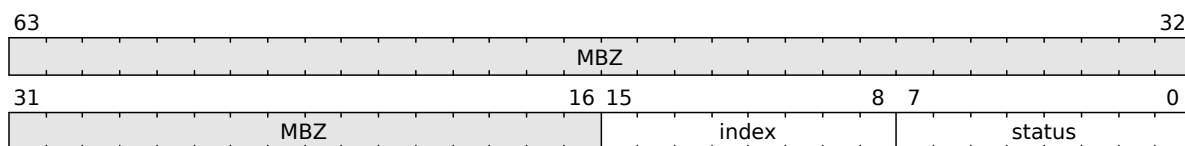
The RmiCommandReturnCode fieldset is a [concrete type](#).

The width of the RmiCommandReturnCode fieldset is 64 bits.

See also:

- [Chapter B1 Commands](#)

The fields of the RmiCommandReturnCode fieldset are shown in the following diagram.



The fields of the RmiCommandReturnCode fieldset are shown in the following table.

Name	Bits	Description	Value
status	7:0	Status of the command	RmiStatusCode
index	15:8	Index which identifies the reason for a command failure	UInt8
	63:16	Reserved	MBZ

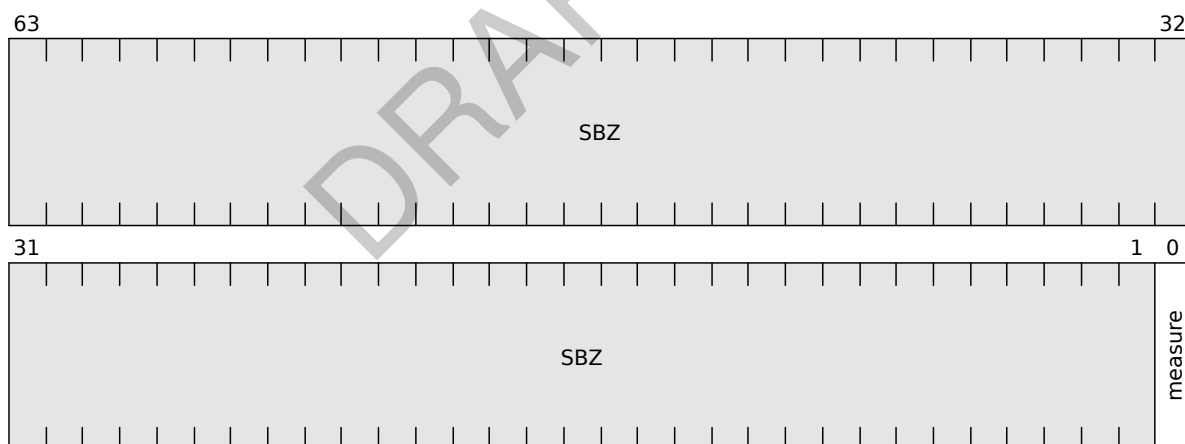
B4.4.4 RmiDataFlags type

The RmiDataFlags fieldset contains flags provided by the Host during DATA Granule creation.

The RmiDataFlags fieldset is a [concrete type](#).

The width of the RmiDataFlags fieldset is 64 bits.

The fields of the RmiDataFlags fieldset are shown in the following diagram.



The fields of the RmiDataFlags fieldset are shown in the following table.

Name	Bits	Description	Value
measure	0	Whether to measure DATA Granule contents	RmiDataMeasureContent
	63:1	Reserved	SBZ

B4.4.5 RmiDataMeasureContent type

The RmiDataMeasureContent enumeration represents whether to measure DATA Granule contents.

The RmiDataMeasureContent enumeration is a [concrete type](#).

The width of the RmiDataMeasureContent enumeration is 1 bits.

The values of the RmiDataMeasureContent enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_NO_MEASURE_CONTENT	Do not measure DATA Granule contents.
1	RMI_MEASURE_CONTENT	Measure DATA Granule contents.

The RmiDataMeasureContent enumeration is used in the following types:

- [RmiDataFlags](#)

B4.4.6 RmiDevCommData type

The RmiDevCommData structure contains data structure shared between Host and RMM for device communication.

The RmiDevCommData structure is a [concrete type](#).

The width of the RmiDevCommData structure is 4096 (0x1000) bytes.

The members of the RmiDevCommData structure are shown in the following table.

Name	Byte offset	Type	Description
enter	0x0	RmiDevCommEnter	Entry information
exit	0x800	RmiDevCommExit	Exit information

Unused bits of the RmiDevCommData structure SBZ.

B4.4.7 RmiDevCommEnter type

The RmiDevCommEnter structure contains data passed from the Host to the RMM during device communication.

The RmiDevCommEnter structure is a [concrete type](#).

The width of the RmiDevCommEnter structure is 256 (0x100) bytes.

See also:

- [A9.2.5.2 Device communication enter data structure](#)

The members of the RmiDevCommEnter structure are shown in the following table.

Name	Byte offset	Type	Description
status	0x0	RmiDevCommStatus	Status of device transaction
req_addr	0x8	Address	Address of request buffer
resp_addr	0x10	Address	Address of response buffer
resp_len	0x18	UInt64	Amount of valid data in response buffer in bytes

Unused bits of the RmiDevCommEnter structure SBZ.

The RmiDevCommEnter structure is used in the following types:

- [RmiDevCommData](#)

B4.4.8 RmiDevCommExit type

The RmiDevCommExit structure contains data passed from the RMM to the Host during device communication.

The RmiDevCommExit structure is a [concrete type](#).

The width of the RmiDevCommExit structure is 256 (0x100) bytes.

See also:

- [A9.2.5.1 Device communication exit data structure](#)

The members of the RmiDevCommExit structure are shown in the following table.

Name	Byte offset	Type	Description
flags	0x0	RmiDevCommExitFlags	Flags indicating action(s) which the Host is requested to perform
cache_offset	0x8	UInt64	If flags.cache is true, offset in the device response buffer to the start of data to be cached, in bytes
cache_len	0x10	UInt64	If flags.cache is true, amount of data to be cached, in bytes
protocol	0x18	RmiDevCommProtocol	If flags.send is true, protocol to use
req_len	0x20	UInt64	If flags.send is true, amount of valid data in request buffer in bytes
timeout	0x28	UInt64	If flags.wait is true, amount of time to wait for device response in milliseconds

Unused bits of the RmiDevCommExit structure MBZ.

The RmiDevCommExit structure is used in the following types:

- [RmiDevCommData](#)

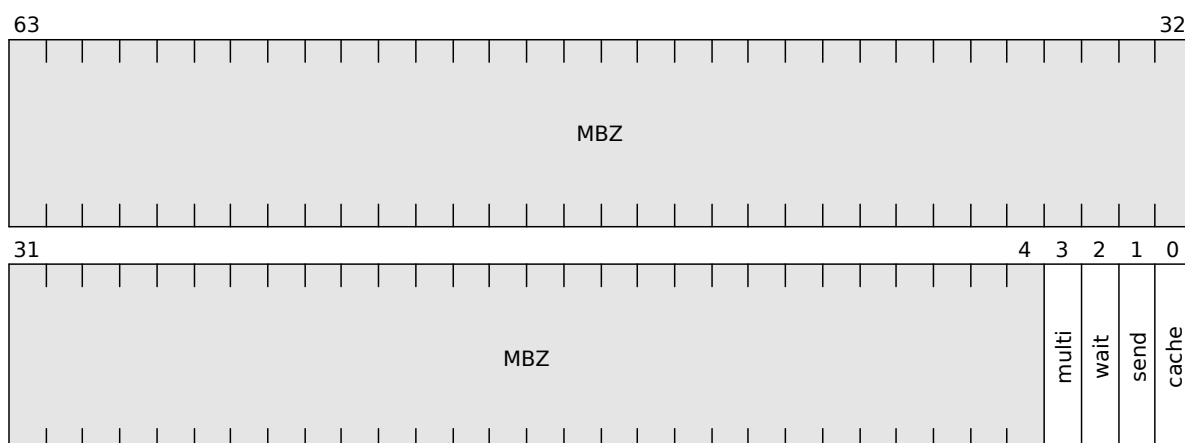
B4.4.9 RmiDevCommExitFlags type

The RmiDevCommExitFlags fieldset contains flags provided by the RMM during a device transaction.

The RmiDevCommExitFlags fieldset is a [concrete type](#).

The width of the RmiDevCommExitFlags fieldset is 64 bits.

The fields of the RmiDevCommExitFlags fieldset are shown in the following diagram.



The fields of the RmiDevCommExitFlags fieldset are shown in the following table.

Name	Bits	Description	Value
cache	0	Whether the Host is requested to cache data from the device response buffer	RmiBoolean
send	1	Whether the Host is requested to send data from the device request buffer to the device	RmiBoolean
wait	2	Whether the RMM is waiting for a response from the device	RmiBoolean
multi	3	Whether the device transaction contains more than one (device request, device response) tuple	RmiBoolean
	63:4	Reserved	MBZ

The RmiDevCommExitFlags fieldset is used in the following types:

- [RmiDevCommExit](#)

B4.4.10 RmiDevCommProtocol type

The RmiDevCommProtocol enumeration represents protocol used for device communication.

The RmiDevCommProtocol enumeration is a [concrete type](#).

The width of the RmiDevCommProtocol enumeration is 8 bits.

The values of the RmiDevCommProtocol enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_PROTOCOL_SPDM	SPDM See Security Protocol and Data Model (SPDM) [19]
1	RMI_PROTOCOL_SECURE_SPDM	Secure SPDM See Security Protocol and Data Model (SPDM) [19]

Unused encodings for the RmiDevCommProtocol enumeration are reserved for use by future versions of this specification.

The RmiDevCommProtocol enumeration is used in the following types:

- [RmiDevCommExit](#)

B4.4.11 RmiDevCommStatus type

The RmiDevCommStatus enumeration represents status passed from the Host to the RMM during device communication.

The RmiDevCommStatus enumeration is a [concrete type](#).

The width of the RmiDevCommStatus enumeration is 8 bits.

The values of the RmiDevCommStatus enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_DEV_COMM_NONE	No device response has been received from the device. Either: <ul style="list-style-type: none">• The device transaction is PENDING, or• The device transaction is ACTIVE and no device response has been received from the device.
1	RMI_DEV_COMM_RESPONSE	The device transaction is ACTIVE and a device response has been received from the device.
2	RMI_DEV_COMM_ERROR	Either: <ul style="list-style-type: none">• The device did not provide a device response within the expected time period, or• The device indicated an error.

Unused encodings for the RmiDevCommStatus enumeration are reserved for use by future versions of this specification.

The RmiDevCommStatus enumeration is used in the following types:

- [RmiDevCommEnter](#)

B4.4.12 RmiEmulatedMmio type

The RmiEmulatedMmio enumeration represents whether the host has completed emulation for an Emulatable Abort.

The RmiEmulatedMmio enumeration is a [concrete type](#).

The width of the RmiEmulatedMmio enumeration is 1 bits.

The values of the RmiEmulatedMmio enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_NOT_EMULATED_MMIO	Host has not completed emulation for an Emulatable Abort.
1	RMI_EMULATED_MMIO	Host has completed emulation for an Emulatable Abort.

The RmiEmulatedMmio enumeration is used in the following types:

- [RmiRecEnterFlags](#)

B4.4.13 RmiFeature type

The RmiFeature enumeration represents whether a feature is supported or enabled.

The RmiFeature enumeration is a [concrete type](#).

The width of the RmiFeature enumeration is 1 bits.

The values of the RmiFeature enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_FEATURE_FALSE	<ul style="list-style-type: none"> During discovery: Feature is not supported. During selection: Feature is not enabled.
1	RMI_FEATURE_TRUE	<ul style="list-style-type: none"> During discovery: Feature is supported. During selection: Feature is enabled.

The RmiFeature enumeration is used in the following types:

- [RmiRealmFlags0](#)
- [RmiVdevFlags](#)
- [RmiFeatureRegister0](#)
- [RmiRealmFlags1](#)

B4.4.14 RmiFeatureRegister0 type

The RmiFeatureRegister0 fieldset contains RMI feature register 0.

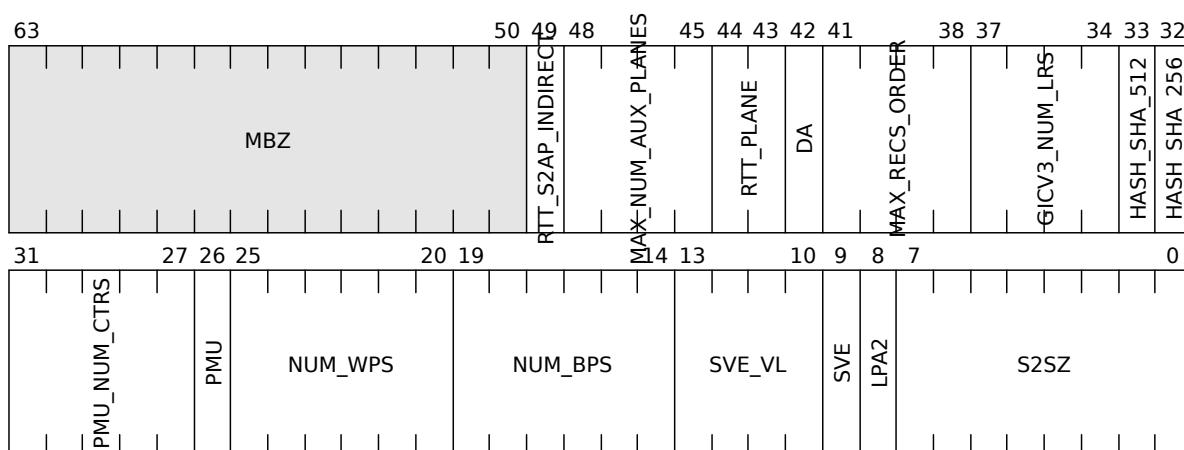
The RmiFeatureRegister0 fieldset is a [concrete type](#).

The width of the RmiFeatureRegister0 fieldset is 64 bits.

See also:

- [Chapter A3 Feature discovery and configuration](#)
- [B4.3.6 RMI_FEATURES command](#)

The fields of the RmiFeatureRegister0 fieldset are shown in the following diagram.



The fields of the RmiFeatureRegister0 fieldset are shown in the following table.

Name	Bits	Description	Value
S2SZ	7:0	Maximum Realm IPA width supported by the RMM. Specifies the input address size for stage 2 translation to be 2^{S2SZ} . Note this format expresses the IPA width directly and is therefore different from the <code>VTTCR_EL2.T0SZ</code> encoding.	UInt8
LPA2	8	Whether LPA2 is supported.	RmiFeature
SVE	9	Whether SVE is supported.	RmiFeature
SVE_VL	13:10	Maximum SVE vector length supported by the RMM. The effective vector length supported by the RMM is $(\text{SVE_VL} + 1) * 128$, similar to the value of <code>ZCR_ELx.LEN</code> .	UInt4
NUM_BPS	19:14	Number of breakpoints available, minus one. The value 0 is reserved.	UInt6
NUM_WPS	25:20	Number of watchpoints available, minus one. The value 0 is reserved.	UInt6
PMU	26	Whether PMU is supported	RmiFeature
PMU_NUM_CTRS	31:27	Number of PMU counters available	UInt5
HASH_SHA_256	32	Whether SHA-256 is supported	RmiFeature
HASH_SHA_512	33	Whether SHA-512 is supported	RmiFeature
GICV3_NUM_LRS	37:34	Number of GICv3 List Registers which are available, minus one.	UInt4
MAX_RECS_ORDER	41:38	Order of the maximum number of RECs which can be created per Realm. The maximum number of RECs is computed as follows: $\text{MAX_RECS} = (2^{\text{MAX_RECS_ORDER}}) - 1$	UInt4
DA	42	Whether Realm device assignment is supported	RmiFeature
RTT_PLANE	44:43	RTT usage models supported for multi-Plane Realms. If only a single Plane is supported (that is, <code>MAX_NUM_AUX_PLANES</code> is 0), this field is RES0.	RmiRttPlaneFeature
MAX_NUM_AUX_PLANES	48:45	Maximum number of auxiliary Planes	UInt4
RTT_S2AP_INDIRECT	49	Whether the Realm uses S2AP indirect encoding	RmiFeature
	63:50	Reserved	MBZ

B4.4.15 RmiFeatureRegister1 type

The RmiFeatureRegister1 fieldset contains RMI feature register 1.

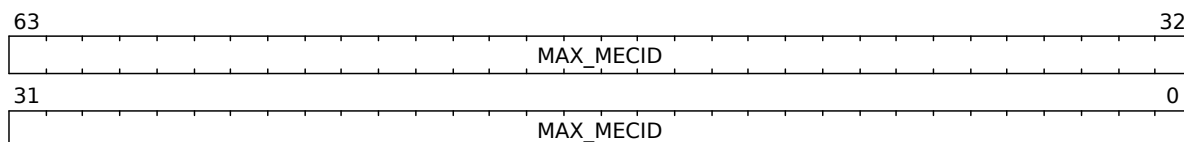
The RmiFeatureRegister1 fieldset is a [concrete type](#).

The width of the RmiFeatureRegister1 fieldset is 64 bits.

See also:

- [Chapter A3 Feature discovery and configuration](#)
- [Chapter A11 Realm memory encryption](#)
- [B4.3.6 RMI_FEATURES command](#)

The fields of the RmiFeatureRegister1 fieldset are shown in the following diagram.



The fields of the RmiFeatureRegister1 fieldset are shown in the following table.

Name	Bits	Description	Value
MAX_MECID	63:0	Maximum MECID.	Bits64

B4.4.16 RmiHashAlgorithm type

The RmiHashAlgorithm enumeration represents hash algorithm.

The RmiHashAlgorithm enumeration is a [concrete type](#).

The width of the RmiHashAlgorithm enumeration is 8 bits.

The values of the RmiHashAlgorithm enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_HASH_SHA_256	SHA-256 (Secure Hash Standard (SHS) [20])
1	RMI_HASH_SHA_512	SHA-512 (Secure Hash Standard (SHS) [20])

Unused encodings for the RmiHashAlgorithm enumeration are reserved for use by future versions of this specification.

The RmiHashAlgorithm enumeration is used in the following types:

- [RmiRealmParams](#)
- [RmiPdevParams](#)

B4.4.17 RmiInjectSea type

The RmiInjectSea enumeration represents whether to inject a Synchronous External Abort into the Realm.

The RmiInjectSea enumeration is a [concrete type](#).

The width of the RmiInjectSea enumeration is 1 bits.

The values of the RmiInjectSea enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_NO_INJECT_SEA	Do not inject an SEA into the Realm.

Encoding	Name	Description
1	RMI_INJECT_SEA	Inject an SEA into the Realm.

The RmiInjectSea enumeration is used in the following types:

- [RmiRecEnterFlags](#)

B4.4.18 RmiInterfaceVersion type

The RmiInterfaceVersion fieldset contains an RMI interface version.

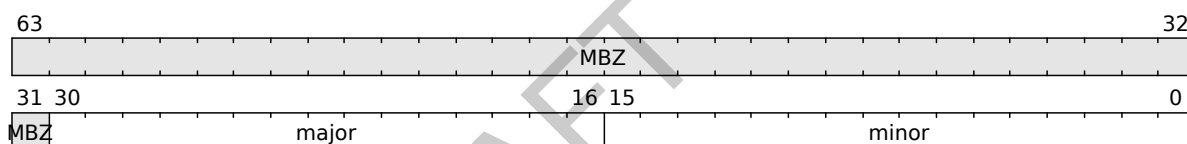
The RmiInterfaceVersion fieldset is a [concrete type](#).

The width of the RmiInterfaceVersion fieldset is 64 bits.

See also:

- [B4.1 RMI version](#)
- [B4.3.56 RMI_VERSION command](#)

The fields of the RmiInterfaceVersion fieldset are shown in the following diagram.



The fields of the RmiInterfaceVersion fieldset are shown in the following table.

Name	Bits	Description	Value
minor	15:0	Interface minor version number (the value y in interface version $x.y$)	UInt16
major	30:16	Interface major version number (the value x in interface version $x.y$)	UInt15
	63:31	Reserved	MBZ

B4.4.19 RmiLfaPolicy type

The RmiLfaPolicy enumeration represents a Live Firmware Activation policy.

The RmiLfaPolicy enumeration is a [concrete type](#).

The width of the RmiLfaPolicy enumeration is 2 bits.

See also:

- [A3.13 Live Firmware Activation](#)

The values of the RmiLfaPolicy enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_LFA_DISALLOW	LFA is not permitted.

Encoding	Name	Description
1	RMI_LFA_ALLOW	LFA is permitted.

Unused encodings for the RmiLfaPolicy enumeration are reserved for use by future versions of this specification.

The RmiLfaPolicy enumeration is used in the following types:

- [RmiRealmFlags0](#)

B4.4.20 RmiPdevEvent type

The RmiPdevEvent enumeration represents physical device event.

The RmiPdevEvent enumeration is a [concrete type](#).

The width of the RmiPdevEvent enumeration is 8 bits.

The values of the RmiPdevEvent enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_IDE_KEY_REFRESH	IDE key refresh.

Unused encodings for the RmiPdevEvent enumeration are reserved for use by future versions of this specification.

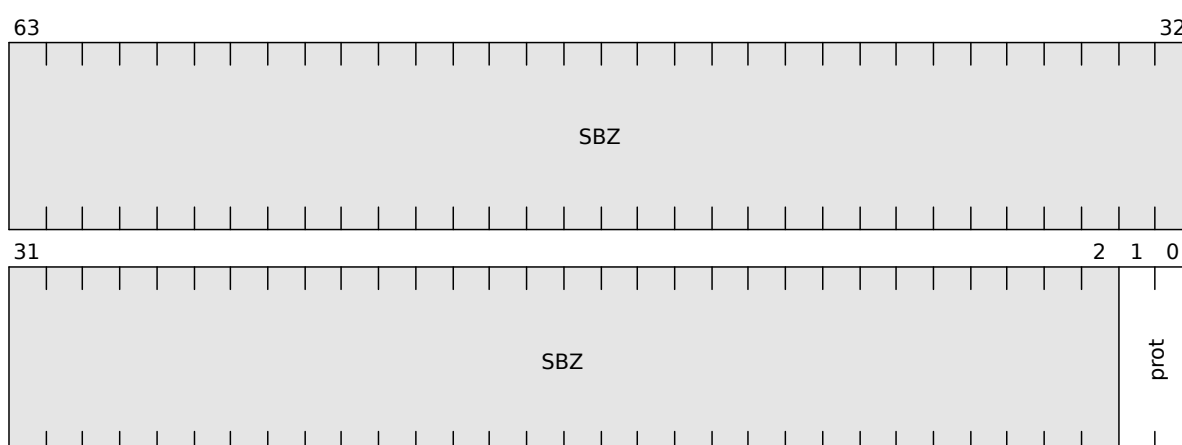
B4.4.21 RmiPdevFlags type

The RmiPdevFlags fieldset contains flags provided by the Host during PDEV creation.

The RmiPdevFlags fieldset is a [concrete type](#).

The width of the RmiPdevFlags fieldset is 64 bits.

The fields of the RmiPdevFlags fieldset are shown in the following diagram.



The fields of the RmiPdevFlags fieldset are shown in the following table.

Name	Bits	Description	Value
prot	1:0	Configuration of protection between system and device	RmiPdevProtection
	63:2	Reserved	SBZ

The RmiPdevFlags fieldset is used in the following types:

- [RmiPdevParams](#)

B4.4.22 RmiPdevParams type

The RmiPdevParams structure contains parameters provided by the Host during PDEV creation.

The RmiPdevParams structure is a [concrete type](#).

The width of the RmiPdevParams structure is 4096 (0x1000) bytes.

The members of the RmiPdevParams structure are shown in the following table.

Name	Byte offset	Type	Description
flags	0x0	RmiPdevFlags	Flags
pdev_id	0x8	Bits64	Physical device identifier For a PCIe device: <ul style="list-style-type: none"> • This is the PCIe routing identifier of the device DOE mailbox which supports CMA. • The value is in PCI BDF format.
segment_id	0x10	Bits8	Segment identifier PCIe Segment identifier to be used in IDE address association.
ecam_addr	0x18	Address	ECAM base address of the PCIe configuration space.
root_id	0x20	Bits16	Root Port identifier Physical PCIe routing identifier of the Root Port to which the endpoint is connected. The value is in PCI BDF format.
cert_id	0x28	UInt64	Certificate identifier
rid_base	0x30	Bits16	Base of requester ID range (inclusive). The value is in PCI BDF format.
rid_top	0x38	Bits16	Top of requester ID range (exclusive). The value is in PCI BDF format.
hash_algo	0x40	RmiHashAlgorithm	Algorithm used to generate device digests
num_aux	0x48	UInt64	Number of auxiliary Granules
ide_sid	0x50	UInt64	IDE stream ID
ncoh_num_addr_ranges	0x58	UInt64	Number of device non-coherent address ranges
coh_num_addr_ranges	0x60	UInt64	Number of device coherent address ranges.

Name	Byte offset	Type	Description
aux[32]	0x100	Address	Addresses of auxiliary Granules
ncoh_addr_range[16]	0x200	RmiAddressRange	Device non-coherent address range
coh_addr_range[4]	0x300	RmiAddressRange	Device coherent address range

Unused bits of the RmiPdevParams structure SBZ.

B4.4.23 RmiPdevProtection type

The RmiPdevProtection enumeration represents protection between system and device.

The RmiPdevProtection enumeration is a [concrete type](#).

The width of the RmiPdevProtection enumeration is 2 bits.

See also:

- [A9.1.1 Device protection types](#)

The values of the RmiPdevProtection enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_PDEV_IOCOH_E2E_IDE	IO-coherent device with end-to-end protection provided by IDE.
1	RMI_PDEV_IOCOH_E2E_SYS	IO-coherent device with end-to-end protection provided by system construction.
2	RMI_PDEV_FCOH_E2E_IDE	Fully-coherent device with end-to-end protection provided by IDE.
3	RMI_PDEV_FCOH_E2E_SYS	Fully-coherent device with end-to-end protection provided by system construction.

The RmiPdevProtection enumeration is used in the following types:

- [RmiPdevFlags](#)

B4.4.24 RmiPdevState type

The RmiPdevState enumeration represents the state of a PDEV.

The RmiPdevState enumeration is a [concrete type](#).

The width of the RmiPdevState enumeration is 8 bits.

The values of the RmiPdevState enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_PDEV_NEW	Initial state of the device.
1	RMI_PDEV_NEEDS_KEY	RMM needs device public key.
2	RMI_PDEV_HAS_KEY	RMM has device public key.

Encoding	Name	Description
3	RMI_PDEV_READY	Secure connection between the RMM and the device has been established. Physical link between the device and memory is secured. Ready for creation of VDEV instances.
4	RMI_PDEV_COMMUNICATING	The RMM is communicating with the device.
5	RMI_PDEV_STOPPING	The RMM is communicating with the device to terminate the secure connection between the RMM and the device.
6	RMI_PDEV_STOPPED	Secure connection between the RMM and the device has been terminated.
7	RMI_PDEV_ERROR	Device has reported a fatal error.

Unused encodings for the RmiPdevState enumeration are reserved for use by future versions of this specification.

B4.4.25 RmiPmuOverflowStatus type

The RmiPmuOverflowStatus enumeration represents PMU overflow status.

The RmiPmuOverflowStatus enumeration is a [concrete type](#).

The width of the RmiPmuOverflowStatus enumeration is 8 bits.

The values of the RmiPmuOverflowStatus enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_PMU_OVERFLOW_NOT_ACTIVE	PMU overflow is not active.
1	RMI_PMU_OVERFLOW_ACTIVE	PMU overflow is active.

Unused encodings for the RmiPmuOverflowStatus enumeration are reserved for use by future versions of this specification.

The RmiPmuOverflowStatus enumeration is used in the following types:

- [RmiRecExit](#)

B4.4.26 RmiPublicKeyParams type

The RmiPublicKeyParams structure contains public key parameters.

The RmiPublicKeyParams structure is a [concrete type](#).

The width of the RmiPublicKeyParams structure is 4096 (0x1000) bytes.

The members of the RmiPublicKeyParams structure are shown in the following table.

Name	Byte offset	Type	Description
key[1024]	0x0	Bits8	Key data
metadata[1024]	0x400	Bits8	Key metadata
key_len	0x800	UInt64	Length of key data in bytes

Name	Byte offset	Type	Description
metadata_len	0x808	UInt64	Length of key metadata in bytes
algo	0x810	RmiSignatureAlgorithm	Signature algorithm

Unused bits of the RmiPublicKeyParams structure SBZ.

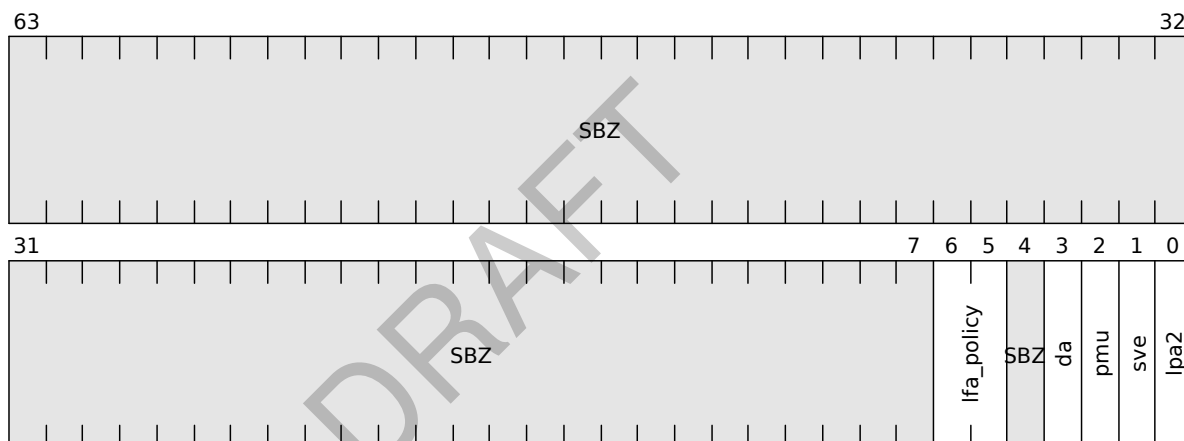
B4.4.27 RmiRealmFlags0 type

The RmiRealmFlags0 fieldset contains flags provided by the Host during Realm creation, which are reflected in Realm Initial Measurement.

The RmiRealmFlags0 fieldset is a [concrete type](#).

The width of the RmiRealmFlags0 fieldset is 64 bits.

The fields of the RmiRealmFlags0 fieldset are shown in the following diagram.



The fields of the RmiRealmFlags0 fieldset are shown in the following table.

Name	Bits	Description	Value
lpa2	0	Whether LPA2 is enabled	RmiFeature
sve	1	Whether SVE is enabled	RmiFeature
pmu	2	Whether PMU is enabled	RmiFeature
da	3	Whether Realm device assignment is enabled	RmiFeature
	4	Reserved	SBZ
lfa_policy	6:5	Live Firmware Activation policy for components within the Realm's TCB	RmiLfaPolicy
	63:7	Reserved	SBZ

The RmiRealmFlags0 fieldset is used in the following types:

- [RmiRealmParams](#)

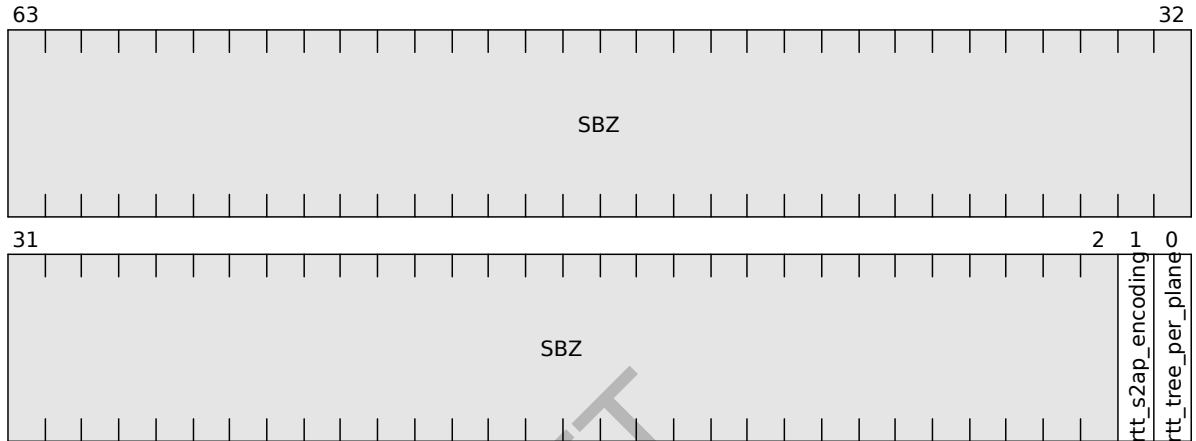
B4.4.28 RmiRealmFlags1 type

The RmiRealmFlags1 fieldset contains flags provided by the Host during Realm creation, which are not reflected in Realm Initial Measurement.

The RmiRealmFlags1 fieldset is a [concrete type](#).

The width of the RmiRealmFlags1 fieldset is 64 bits.

The fields of the RmiRealmFlags1 fieldset are shown in the following diagram.



The fields of the RmiRealmFlags1 fieldset are shown in the following table.

Name	Bits	Description	Value
rtt_tree_per_plane	0	RMI_FEATURE_FALSE: all Planes share a single RTT tree RMI_FEATURE_TRUE: each Plane has a separate RTT tree If the Realm has no auxiliary Planes then this field is ignored.	RmiFeature
rtt_s2ap_encoding	1	S2AP encoding	RmiRttS2APEncoding
	63:2	Reserved	SBZ

The RmiRealmFlags1 fieldset is used in the following types:

- [RmiRealmParams](#)

B4.4.29 RmiRealmParams type

The RmiRealmParams structure contains parameters provided by the Host during Realm creation.

The RmiRealmParams structure is a [concrete type](#).

The width of the RmiRealmParams structure is 4096 (0x1000) bytes.

See also:

- [A2.1.6 Realm parameters](#)
- [B4.3.25 RMI_REALM_CREATE command](#)

The members of the RmiRealmParams structure are shown in the following table.

Name	Byte offset	Type	Description
flags0	0x0	RmiRealmFlags0	Flags
s2sz	0x8	UInt8	IPA width. Specifies the input address size for stage 2 translation to be 2^{s2sz} . Note this format expresses the IPA width directly and is therefore different from the <code>VTCR_EL2.T0SZ</code> encoding.
sve_vl	0x10	UInt8	SVE vector length. The effective vector length requested is $(\text{sve_vl} + 1) * 128$, similar to the value of <code>ZCR_ELx.LEN</code> .
num_bps	0x18	UInt8	Number of breakpoints, minus one. The value 0 is reserved.
num_wps	0x20	UInt8	Number of watchpoints, minus one. The value 0 is reserved.
pmu_num_ctrs	0x28	UInt8	Number of PMU counters
hash_algo	0x30	RmiHashAlgorithm	Algorithm used to measure the initial state of the Realm
num_aux_planes	0x38	UInt64	Number of auxiliary Planes
rpv	0x400	Bits512	Realm Personalization Value
vmid	0x800	Bits16	Primary Virtual Machine Identifier
rtt_base	0x808	Address	Base address of primary RTT
rtt_level_start	0x810	Int64	RTT starting level
rtt_num_start	0x818	UInt32	Number of starting level RTTs
flags1	0x820	RmiRealmFlags1	Flags
mecid	0x828	Bits64	MECID
aux_vmid[3]	0xf00	Bits16	Auxiliary Virtual Machine Identifiers
aux_rtt_base[3]	0xf80	Address	Base address of auxiliary RTTs

Unused bits of the `RmiRealmParams` structure SBZ.

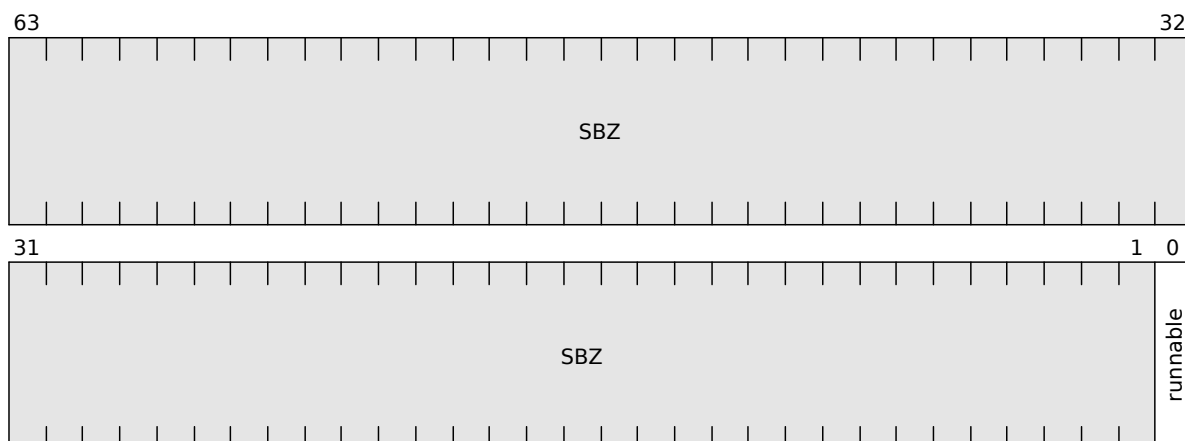
B4.4.30 RmiRecCreateFlags type

The `RmiRecCreateFlags` fieldset contains flags provided by the Host during REC creation.

The `RmiRecCreateFlags` fieldset is a [concrete type](#).

The width of the `RmiRecCreateFlags` fieldset is 64 bits.

The fields of the `RmiRecCreateFlags` fieldset are shown in the following diagram.



The fields of the RmiRecCreateFlags fieldset are shown in the following table.

Name	Bits	Description	Value
runnable	0	Whether REC is eligible for execution	RmiRecRunnable
	63:1	Reserved	SBZ

The RmiRecCreateFlags fieldset is used in the following types:

- [RmiRecParams](#)

B4.4.31 RmiRecEnter type

The RmiRecEnter structure contains data passed from the Host to the RMM on REC entry.

The RmiRecEnter structure is a [concrete type](#).

The width of the RmiRecEnter structure is 2048 (0x800) bytes.

See also:

- [A4.2.1 RmiRecEnter object](#)
- [B4.3.30 RMI_REC_ENTER command](#)
- [B4.4.33 RmiRecExit type](#)

The members of the RmiRecEnter structure are shown in the following table.

Name	Byte offset	Type	Description
flags	0x0	RmiRecEnterFlags	Flags
gprs[31]	0x200	Bits64	Registers
gicv3_hcr	0x300	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0x308	Bits64	GICv3 List Register values

Unused bits of the RmiRecEnter structure SBZ.

The RmiRecEnter structure is used in the following types:

- [RmiRecRun](#)

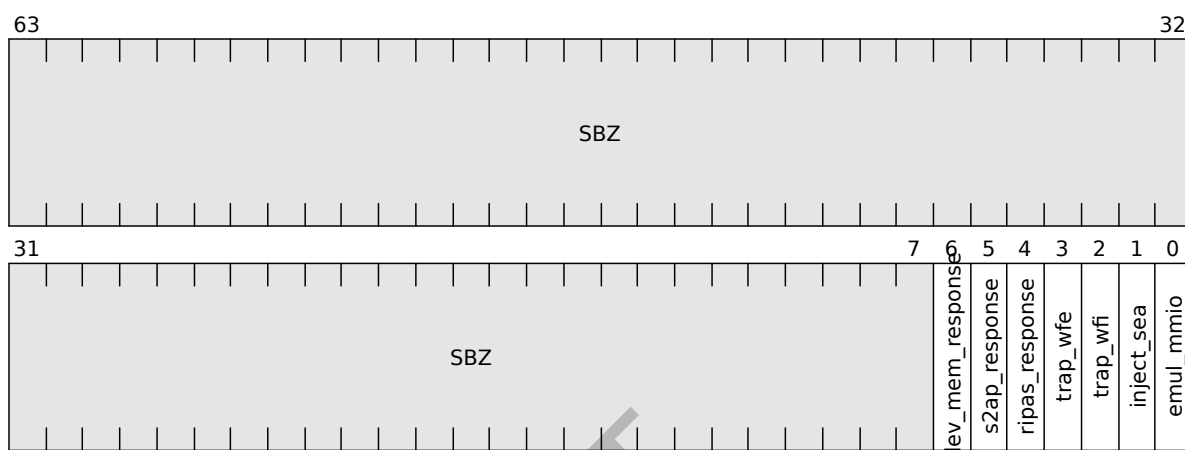
B4.4.32 RmiRecEnterFlags type

The RmiRecEnterFlags fieldset contains flags provided by the Host during REC entry.

The RmiRecEnterFlags fieldset is a [concrete type](#).

The width of the RmiRecEnterFlags fieldset is 64 bits.

The fields of the RmiRecEnterFlags fieldset are shown in the following diagram.



The fields of the RmiRecEnterFlags fieldset are shown in the following table.

Name	Bits	Description	Value
emul_mmio	0	Whether the host has completed emulation for an Emulatable Data Abort	RmiEmulatedMmio
inject_sea	1	Whether to inject a Synchronous External Abort into the Realm.	RmiInjectSea
trap_wfi	2	Whether to trap WFI execution by the Realm.	RmiTrap
trap_wfe	3	Whether to trap WFE execution by the Realm.	RmiTrap
ripas_response	4	Host response to RIPAS change request.	RmiResponse
s2ap_response	5	Host response to S2AP change request.	RmiResponse
dev_mem_response	6	Host response to device memory mapping validation request.	RmiResponse
	63:7	Reserved	SBZ

The RmiRecEnterFlags fieldset is used in the following types:

- [RmiRecEnter](#)

B4.4.33 RmiRecExit type

The RmiRecExit structure contains data passed from the RMM to the Host on REC exit.

The RmiRecExit structure is a [concrete type](#).

The width of the RmiRecExit structure is 2048 (0x800) bytes.

See also:

- [A4.3.1 RmiRecExit object](#)
- [B4.3.30 RMI_REC_ENTER command](#)
- [B4.4.31 RmiRecEnter type](#)

The members of the RmiRecExit structure are shown in the following table.

Name	Byte offset	Type	Description
exit_reason	0x0	RmiRecExitReason	Exit reason
esr	0x100	Bits64	Exception Syndrome Register
far	0x108	Bits64	Fault Address Register
hpfar	0x110	Bits64	Hypervisor IPA Fault Address register
rtt_tree	0x118	UInt64	Index of RTT tree active at time of the exit
rtt_level	0x120	Int64	Level of requested RTT
gprs[31]	0x200	Bits64	Registers
gicv3_hcr	0x300	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0x308	Bits64	GICv3 List Register values
gicv3_misr	0x388	Bits64	GICv3 Maintenance Interrupt State Register value
gicv3_vmcr	0x390	Bits64	GICv3 Virtual Machine Control Register value
cntp_ctl	0x400	Bits64	Counter-timer Physical Timer Control Register value
cntp_cval	0x408	Bits64	Counter-timer Physical Timer CompareValue Register value
cntv_ctl	0x410	Bits64	Counter-timer Virtual Timer Control Register value
cntv_cval	0x418	Bits64	Counter-timer Virtual Timer CompareValue Register value
ripas_base	0x500	Bits64	Base IPA of target region for pending RIPAS change
ripas_top	0x508	Bits64	Top IPA of target region for pending RIPAS change
ripas_value	0x510	RmiRipas	RIPAS value of pending RIPAS change
s2ap_base	0x520	Bits64	Base IPA of target region for pending S2AP change
s2ap_top	0x528	Bits64	Top IPA of target region for pending S2AP change
vdev_id	0x530	Bits64	Virtual device ID
imm	0x600	Bits16	Host call immediate value
plane	0x608	UInt64	Plane index
vdev	0x610	Address	VDEV which triggered REC exit due to VDEV communication

Name	Byte offset	Type	Description
vdev_action	0x618	RmiVdevAction	Action which triggered REC exit due to VDEV communication
dev_mem_base	0x620	Bits64	Base IPA of target region for device memory mapping validation
dev_mem_top	0x628	Bits64	Top IPA of target region for device memory mapping validation
dev_mem_pa	0x630	Address	Base PA of device memory region
pmu_ovf_status	0x700	RmiPmuOverflowStatus	PMU overflow status

Unused bits of the RmiRecExit structure MBZ.

The RmiRecExit structure is used in the following types:

- [RmiRecRun](#)

B4.4.34 RmiRecExitReason type

The RmiRecExitReason enumeration represents the reason for a REC exit.

The RmiRecExitReason enumeration is a [concrete type](#).

The width of the RmiRecExitReason enumeration is 8 bits.

The values of the RmiRecExitReason enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_EXIT_SYNC	REC exit due to synchronous exception
1	RMI_EXIT_IRQ	REC exit due to IRQ
2	RMI_EXIT_FIQ	REC exit due to FIQ
3	RMI_EXIT_PSCI	REC exit due to PSCI
4	RMI_EXIT_RIPAS_CHANGE	REC exit due to RIPAS change pending
5	RMI_EXIT_HOST_CALL	REC exit due to Host call
6	RMI_EXIT_SERROR	REC exit due to SError
7	RMI_EXIT_S2AP_CHANGE	REC exit due to S2AP change pending
8	RMI_EXIT_VDEV_REQUEST	REC exit due to VDEV request
9	RMI_EXIT_VDEV_COMM	REC exit due to VDEV communication
10	RMI_EXIT_DEV_MEM_MAP	REC exit due to device memory mapping validation

Unused encodings for the RmiRecExitReason enumeration are reserved for use by future versions of this specification.

The RmiRecExitReason enumeration is used in the following types:

- [RmiRecExit](#)

B4.4.35 RmiRecMpidr type

The RmiRecMpidr fieldset contains MPIDR value which identifies a REC.

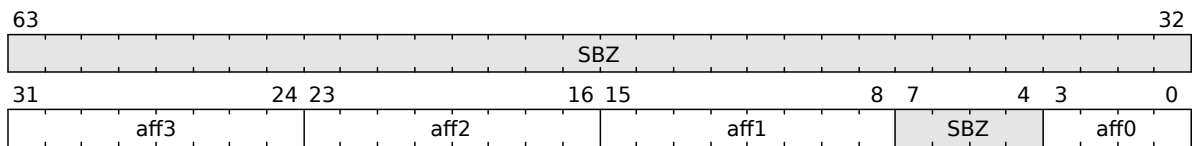
The RmiRecMpidr fieldset is a [concrete type](#).

The width of the RmiRecMpidr fieldset is 64 bits.

See also:

- [A2.3.3 REC index and MPIDR value](#)
- [B4.3.28 RMI_REC_CREATE command](#)

The fields of the RmiRecMpidr fieldset are shown in the following diagram.



The fields of the RmiRecMpidr fieldset are shown in the following table.

Name	Bits	Description	Value
aff0	3:0	Affinity level 0	Bits4
	7:4	Reserved	SBZ
aff1	15:8	Affinity level 1	Bits8
aff2	23:16	Affinity level 2	Bits8
aff3	31:24	Affinity level 3	Bits8
	63:32	Reserved	SBZ

The RmiRecMpidr fieldset is used in the following types:

- [RmiRecParams](#)

B4.4.36 RmiRecParams type

The RmiRecParams structure contains parameters provided by the Host during REC creation.

The RmiRecParams structure is a [concrete type](#).

The width of the RmiRecParams structure is 4096 (0x1000) bytes.

The number of valid entries in the `aux` array is determined by the return value from the `RMI_REC_AUX_COUNT` command.

See also:

- [B4.3.27 RMI_REC_AUX_COUNT command](#)

The members of the RmiRecParams structure are shown in the following table.

Name	Byte offset	Type	Description
flags	0x0	RmiRecCreateFlags	Flags
mpidr	0x100	RmiRecMpidr	MPIDR of the REC

Name	Byte offset	Type	Description
pc	0x200	Bits64	Program counter
gprs[8]	0x300	Bits64	General-purpose registers
num_aux	0x800	UInt64	Number of auxiliary Granules
aux[16]	0x808	Address	Addresses of auxiliary Granules

Unused bits of the RmiRecParams structure SBZ.

B4.4.37 RmiRecRun type

The RmiRecRun structure contains fields used to share information between RMM and Host during REC entry and REC exit.

The RmiRecRun structure is a [concrete type](#).

The width of the RmiRecRun structure is 4096 (0x1000) bytes.

See also:

- [A4.2.1 RmiRecEnter object](#)
- [A4.3.1 RmiRecExit object](#)
- [B4.3.30 RMI_REC_ENTER command](#)

The members of the RmiRecRun structure are shown in the following table.

Name	Byte offset	Type	Description
enter	0x0	RmiRecEnter	Entry information
exit	0x800	RmiRecExit	Exit information

B4.4.38 RmiRecRunnable type

The RmiRecRunnable enumeration represents whether a REC is eligible for execution.

The RmiRecRunnable enumeration is a [concrete type](#).

The width of the RmiRecRunnable enumeration is 1 bits.

The values of the RmiRecRunnable enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_NOT_RUNNABLE	Not eligible for execution.
1	RMI_RUNNABLE	Eligible for execution.

The RmiRecRunnable enumeration is used in the following types:

- [RmiRecCreateFlags](#)

B4.4.39 RmiResponse type

The RmiResponse enumeration represents whether the Host accepted or rejected a Realm request.

The RmiResponse enumeration is a [concrete type](#).

The width of the RmiResponse enumeration is 1 bits.

The values of the RmiResponse enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_ACCEPT	Host accepted the Realm request.
1	RMI_REJECT	Host rejected the Realm request.

The RmiResponse enumeration is used in the following types:

- [RmiRecEnterFlags](#)

B4.4.40 RmiRipas type

The RmiRipas enumeration represents realm IPA state.

The RmiRipas enumeration is a [concrete type](#).

The width of the RmiRipas enumeration is 8 bits.

The values of the RmiRipas enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_EMPTY	Address where no Realm resources are mapped.
1	RMI_RAM	Address where private code or data owned by the Realm is mapped.
2	RMI_DESTROYED	Address which is inaccessible to the Realm due to an action taken by the Host.
3	RMI_DEV	Address where memory of an assigned Realm device is mapped.

Unused encodings for the RmiRipas enumeration are reserved for use by future versions of this specification.

The RmiRipas enumeration is used in the following types:

- [RmiRecExit](#)

B4.4.41 RmiRttEntryState type

The RmiRttEntryState enumeration represents the state of an RTTE.

The RmiRttEntryState enumeration is a [concrete type](#).

The width of the RmiRttEntryState enumeration is 8 bits.

The values of the RmiRttEntryState enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_UNASSIGNED	This RTTE is not associated with any Granule.
1	RMI_ASSIGNED	The output address of this RTTE points to: <ul style="list-style-type: none">• a DATA Granule, if the input address is a Protected IPA, or• any Granule-aligned address within NS PAS, if the input address is an Unprotected IPA.
2	RMI_TABLE	The output address of this RTTE points to the next-level RTT.
3	RMI_ASSIGNED_DEV	The output address of this RTTE points to an DEV_MAPPED Granule.
4	RMI_AUX_DESTROYED	An auxiliary RTT was destroyed.
5	RMI_ASSIGNED_VSMMU	The output address of this RTTE points to a VSMMU Granule.

Unused encodings for the RmiRttEntryState enumeration are reserved for use by future versions of this specification.

B4.4.42 RmiRttPlaneFeature type

The RmiRttPlaneFeature enumeration represents RTT usage models supported for multi-Plane Realms.

The RmiRttPlaneFeature enumeration is a [concrete type](#).

The width of the RmiRttPlaneFeature enumeration is 2 bits.

See also:

- [A3.11 Support for auxiliary Planes](#)

The values of the RmiRttPlaneFeature enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_RTT_PLANE_AUX	A multi-Plane Realm uses auxiliary RTTs
1	RMI_RTT_PLANE_AUX_SINGLE	A multi-Plane Realm can be configured to either use auxiliary RTTs, or a single RTT
2	RMI_RTT_PLANE_SINGLE	A multi-Plane Realm uses a single RTT

Unused encodings for the RmiRttPlaneFeature enumeration are reserved for use by future versions of this specification.

The RmiRttPlaneFeature enumeration is used in the following types:

- [RmiFeatureRegister0](#)

B4.4.43 RmiRttS2APEncoding type

The RmiRttS2APEncoding enumeration represents S2AP encoding.

The RmiRttS2APEncoding enumeration is a [concrete type](#).

The width of the RmiRttS2APEncoding enumeration is 1 bits.

The values of the RmiRttS2APEncoding enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_S2AP_DIRECT	S2AP direct encoding.
1	RMI_S2AP_INDIRECT	S2AP indirect encoding.

The RmiRttS2APEncoding enumeration is used in the following types:

- [RmiRealmFlags1](#)

B4.4.44 RmiSignatureAlgorithm type

The RmiSignatureAlgorithm enumeration represents signature algorithm.

The RmiSignatureAlgorithm enumeration is a [concrete type](#).

The width of the RmiSignatureAlgorithm enumeration is 8 bits.

The values of the RmiSignatureAlgorithm enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_SIG_RSASSA_3072	SSA-3072 (RSA Cryptography Specifications Version 2.2 [21])
1	RMI_SIG_ECDSA_P256	ECDSA-P256 (Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) [22])
2	RMI_SIG_ECDSA_P384	ECDSA-P384 (Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) [22])

Unused encodings for the RmiSignatureAlgorithm enumeration are reserved for use by future versions of this specification.

The RmiSignatureAlgorithm enumeration is used in the following types:

- [RmiPublicKeyParams](#)

B4.4.45 RmiSmmuAction type

The RmiSmmuAction enumeration represents action required by Host in response to an SMMU interrupt.

The RmiSmmuAction enumeration is a [concrete type](#).

The width of the RmiSmmuAction enumeration is 1 bits.

The values of the RmiSmmuAction enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_SMMU_ACTION_NONE	No action required.
1	RMI_SMMU_ACTION_VIRQ	Inject a virtual interrupt into a Realm.

B4.4.46 RmiSmmuIrq type

The RmiSmmuIrq enumeration represents SMMU IRQ.

The RmiSmmuIrq enumeration is a [concrete type](#).

The width of the RmiSmmuIrq enumeration is 2 bits.

The values of the RmiSmmuIrq enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_SMMU_IRQ_GERROR	GERROR interrupt.
1	RMI_SMMU_IRQ_EVENTQ	EVENTQ interrupt.
2	RMI_SMMU_IRQ_PRIQ	PRIQ interrupt.

Unused encodings for the RmiSmmuIrq enumeration are reserved for use by future versions of this specification.

B4.4.47 RmiStatusCode type

The RmiStatusCode enumeration represents the status of an RMI operation.

The RmiStatusCode enumeration is a [concrete type](#).

The width of the RmiStatusCode enumeration is 8 bits.

See also:

- [B1.3 Command registers](#)
- [B1.5 Command context values](#)

The values of the RmiStatusCode enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_SUCCESS	Command completed successfully
1	RMI_ERROR_INPUT	The value of a command input value caused the command to fail
2	RMI_ERROR_REALM	An attribute of a Realm does not match the expected value
3	RMI_ERROR_REC	An attribute of a REC does not match the expected value
4	RMI_ERROR_RTT	An RTT walk terminated before reaching the target RTT level, or reached an RTTE with an unexpected value
5	RMI_ERROR_NOT_SUPPORTED	The command is not supported
6	RMI_ERROR_DEVICE	An attribute of a device does not match the expected value
7	RMI_ERROR_RTT_AUX	RTTE in an auxiliary RTT contained an unexpected value

Unused encodings for the RmiStatusCode enumeration are reserved for use by future versions of this specification.

The RmiStatusCode enumeration is used in the following types:

- [RmiCommandReturnCode](#)

B4.4.48 RmiTrap type

The RmiTrap enumeration represents whether a trap is enabled.

The RmiTrap enumeration is a [concrete type](#).

The width of the RmiTrap enumeration is 1 bits.

The values of the RmiTrap enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_NO_TRAP	Trap is disabled.
1	RMI_TRAP	Trap is enabled.

The RmiTrap enumeration is used in the following types:

- [RmiRecEnterFlags](#)

B4.4.49 RmiVdevAction type

The RmiVdevAction enumeration represents realm action which triggered REC exit due to VDEV communication.

The RmiVdevAction enumeration is a [concrete type](#).

The width of the RmiVdevAction enumeration is 8 bits.

The values of the RmiVdevAction enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_VDEV_ACTION_GET_INTERFACE_REPORT	Exit triggered by RSI_RDEV_GET_INTERFACE_REPORT
1	RMI_VDEV_ACTION_GET_MEASUREMENTS	Exit triggered by RSI_RDEV_GET_MEASUREMENTS
2	RMI_VDEV_ACTION_LOCK	Exit triggered by RSI_RDEV_LOCK
3	RMI_VDEV_ACTION_START	Exit triggered by RSI_RDEV_START
4	RMI_VDEV_ACTION_STOP	Exit triggered by RSI_RDEV_STOP

Unused encodings for the RmiVdevAction enumeration are reserved for use by future versions of this specification.

The RmiVdevAction enumeration is used in the following types:

- [RmiRecExit](#)

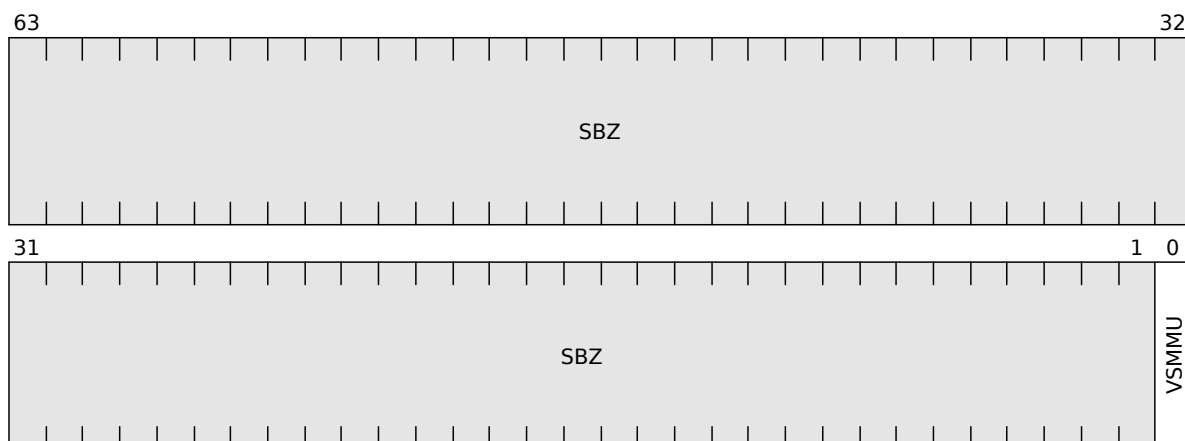
B4.4.50 RmiVdevFlags type

The RmiVdevFlags fieldset contains flags provided by the Host during VDEV creation.

The RmiVdevFlags fieldset is a [concrete type](#).

The width of the RmiVdevFlags fieldset is 64 bits.

The fields of the RmiVdevFlags fieldset are shown in the following diagram.



The fields of the RmiVdevFlags fieldset are shown in the following table.

Name	Bits	Description	Value
VSMMU	0	Whether device uses a VSMMU	RmiFeature
	63:1	Reserved	SBZ

The RmiVdevFlags fieldset is used in the following types:

- [RmiVdevParams](#)

B4.4.51 RmiVdevParams type

The RmiVdevParams structure contains parameters provided by the Host during VDEV creation.

The RmiVdevParams structure is a [concrete type](#).

The width of the RmiVdevParams structure is 4096 (0x1000) bytes.

The members of the RmiVdevParams structure are shown in the following table.

Name	Byte offset	Type	Description
flags	0x0	RmiVdevFlags	Flags
vdev_id	0x8	Bits64	Virtual device identifier For a PCIe device this is the PCIe routing identifier of the virtual endpoint.
tdi_id	0x10	Bits64	TDI identifier
num_aux	0x18	UInt64	Number of auxiliary Granules
vsmmu_addr	0x20	Address	PA of VSMMU. This field is ignored unless flags.VSMMU is RMI_TRUE.
vsid	0x28	Bits64	Virtual Stream Identifier. This field is ignored unless flags.VSMMU is RMI_TRUE.
aux[32]	0x100	Address	Addresses of auxiliary Granules

Unused bits of the RmiVdevParams structure SBZ.

B4.4.52 RmiVdevState type

The RmiVdevState enumeration represents the state of a VDEV.

The RmiVdevState enumeration is a [concrete type](#).

The width of the RmiVdevState enumeration is 8 bits.

The values of the RmiVdevState enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_VDEV_NEW	Initial state of the device.
1	RMI_VDEV_READY	No device transaction is associated with the VDEV.
2	RMI_VDEV_COMMUNICATING	The RMM is communicating with the VDEV.
3	RMI_VDEV_STOPPING	The RMM is communicating with the VDEV to stop the device interface.
4	RMI_VDEV_STOPPED	Device interface is stopped.
5	RMI_VDEV_ERROR	Device interface has reported a fatal error.

Unused encodings for the RmiVdevState enumeration are reserved for use by future versions of this specification.

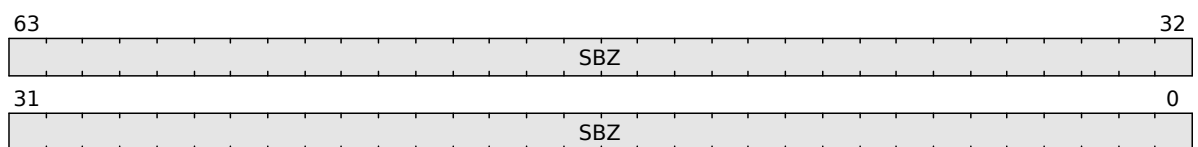
B4.4.53 RmiVsmmuFlags type

The RmiVsmmuFlags fieldset contains flags provided by the Host during PDEV creation.

The RmiVsmmuFlags fieldset is a [concrete type](#).

The width of the RmiVsmmuFlags fieldset is 64 bits.

The fields of the RmiVsmmuFlags fieldset are shown in the following diagram.



The fields of the RmiVsmmuFlags fieldset are shown in the following table.

Name	Bits	Description	Value
	63:0	Reserved	SBZ

The RmiVsmmuFlags fieldset is used in the following types:

- [RmiVsmmuParams](#)

B4.4.54 RmiVsmmuParams type

The RmiVsmmuParams structure contains parameters provided by the Host during VSMMU creation.

The RmiVsmmuParams structure is a [concrete type](#).

The width of the RmiVsmmuParams structure is 4096 (0x1000) bytes.

The members of the RmiVsmmuParams structure are shown in the following table.

Name	Byte offset	Type	Description
flags	0x0	RmiVsmmuFlags	Flags
reg_base	0x8	Address	Base IPA of register base in Realm's Protected IPA space
reg_top	0x10	Address	Top IPA of register base in Realm's Protected IPA space
aidr	0x18	Bits64	SMMU_AIDR register value
idr[7]	0x20	Bits64	SMMU_IDR register values

Unused bits of the RmiVsmmuParams structure SBZ.

DRAFT

Chapter B5

Realm Services Interface

This chapter defines the interface used by Realm software to request services from the RMM.

B5.1 RSI version

R_{QKLGZ} This specification defines version 1.1 of the Realm Services Interface.

See also:

- [Chapter B2 Interface versioning](#)
- [B5.3.26 RSI_VERSION command](#)

B5.2 RSI command return codes

I_{CYQDJ} An RSI command return code indicates whether the command

- succeeded, or
- failed, and the reason for the failure.

I_{DQJSP} If an RSI command succeeds then it returns RSI_SUCCESS.

I_{YMHKC} Multiple failure conditions in an RSI command may return the same return code.

R_{MLBDM} If an input to an RSI command uses an invalid encoding then the command fails and returns RSI_ERROR_INPUT.

Command inputs include registers and in-memory data structures.

Invalid encodings include:

- using a reserved encoding in an enumeration

See also:

- [B5.4.2 RsiCommandReturnCode type](#)

B5.3 RSI commands

The following table summarizes the FIDs of commands in the RSI interface.

FID	Command
0xC4000190	RSI_VERSION
0xC4000191	RSI_FEATURES
0xC4000192	RSI_MEASUREMENT_READ
0xC4000193	RSI_MEASUREMENT_EXTEND
0xC4000194	RSI_ATTESTATION_TOKEN_INIT
0xC4000195	RSI_ATTESTATION_TOKEN_CONTINUE
0xC4000196	RSI_REALM_CONFIG
0xC4000197	RSI_IPA_STATE_SET
0xC4000198	RSI_IPA_STATE_GET
0xC4000199	RSI_HOST_CALL
0xC400019A	RSI_VSMMU_GET_INFO
0xC400019B	RSI_VSMMU_ACTIVATE
0xC400019C	RSI_RDEV_GET_INSTANCE_ID
...	
0xC40001A0	RSI_MEM_GET_PERM_VALUE
0xC40001A1	RSI_MEM_SET_PERM_INDEX
0xC40001A2	RSI_MEM_SET_PERM_VALUE
0xC40001A3	RSI_PLANE_ENTER
0xC40001A4	RSI_RDEV_CONTINUE
0xC40001A5	RSI_RDEV_GET_INFO
0xC40001A6	RSI_RDEV_GET_INTERFACE_REPORT
0xC40001A7	RSI_RDEV_GET_MEASUREMENTS
0xC40001A8	RSI_RDEV_GET_STATE
0xC40001A9	RSI_RDEV_LOCK
0xC40001AA	RSI_RDEV_START
0xC40001AB	RSI_RDEV_STOP
0xC40001AC	RSI_RDEV_VALIDATE_MAPPING
...	
0xC40001AE	RSI_PLANE_REG_READ
0xC40001AF	RSI_PLANE_REG_WRITE

B5.3.1 RSI_ATTESTATION_TOKEN_CONTINUE command

Continue the operation to retrieve an attestation token.

See also:

- [A5.2.4 RSI command access to a Protected IPA](#)
- [A7.2 Realm attestation](#)
- [B5.3.2 RSI_ATTESTATION_TOKEN_INIT command](#)

B5.3.1.1 Interface

B5.3.1.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000195
addr	X1	63:0	Address	IPA of the Granule to which the token will be written
offset	X2	63:0	UInt64	Offset within Granule to start of buffer in bytes
size	X3	63:0	UInt64	Size of buffer in bytes

B5.3.1.1.2 Context

The RSI_ATTESTATION_TOKEN_CONTINUE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rec	RmmRec	CurrentRec()	false	Current REC
walk	RmmRttWalkResult	RttWalk(realm, addr, RMM_RTT_PAGE_LEVEL, RMM_RTT_TREE_PRIMARY)	false	RTT walk result

B5.3.1.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
len	X1	63:0	UInt64	Number of bytes written to buffer

B5.3.1.2 Failure conditions

ID	Condition
addr_align	pre: !AddrIsGranuleAligned(addr) post: result == RSI_ERROR_INPUT

ID	Condition
addr_bound	pre: !AddrIsProtected(addr, realm) post: result == RSI_ERROR_INPUT
addr_empty	pre: walk.rtte.ripas == EMPTY post: result == RSI_ERROR_INPUT
offset_bound	pre: offset >= RMM_GRANULE_SIZE post: result == RSI_ERROR_INPUT
size_overflow	pre: offset + size < offset post: result == RSI_ERROR_INPUT
size_bound	pre: offset + size > RMM_GRANULE_SIZE post: result == RSI_ERROR_INPUT
state	pre: rec.attest_state != ATTEST_IN_PROGRESS post: result == RSI_ERROR_STATE
unknown	pre: Token generation failed for an unknown or IMPDEF reason. post: result == RSI_ERROR_UNKNOWN

B5.3.1.2.1 Failure condition ordering

The RSI_ATTESTATION_TOKEN_CONTINUE command does not have any failure condition orderings.

B5.3.1.3 Success conditions

ID	Condition
incomplete	pre: Token generation is not complete. post: result == RSI_INCOMPLETE
complete	pre: Token generation is complete. post: rec.attest_state == NO_ATTEST_IN_PROGRESS

B5.3.1.4 Footprint

ID	Value
state	rec.attest_state

B5.3.2 RSI_ATTESTATION_TOKEN_INIT command

Initialize the operation to retrieve an attestation token.

See also:

- [A7.2 Realm attestation](#)
- [B5.3.1 RSI_ATTESTATION_TOKEN_CONTINUE command](#)

B5.3.2.1 Interface

B5.3.2.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000194
challenge_0	X1	63:0	Bits64	Doubleword 0 of the challenge value
challenge_1	X2	63:0	Bits64	Doubleword 1 of the challenge value
challenge_2	X3	63:0	Bits64	Doubleword 2 of the challenge value
challenge_3	X4	63:0	Bits64	Doubleword 3 of the challenge value
challenge_4	X5	63:0	Bits64	Doubleword 4 of the challenge value
challenge_5	X6	63:0	Bits64	Doubleword 5 of the challenge value
challenge_6	X7	63:0	Bits64	Doubleword 6 of the challenge value
challenge_7	X8	63:0	Bits64	Doubleword 7 of the challenge value

B5.3.2.1.2 Context

The RSI_ATTESTATION_TOKEN_INIT command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rec	RmmRec	CurrentRec()	false	Current REC

B5.3.2.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
size	X1	63:0	UInt64	Upper bound on attestation token size in bytes

B5.3.2.2 Failure conditions

The RSI_ATTESTATION_TOKEN_INIT command does not have any failure conditions.

B5.3.2.3 Success conditions

ID	Condition
state	<code>rec.attest_state == ATTEST_IN_PROGRESS</code>
challenge	<code>rec.attest_challenge == [</code> <code>challenge_0,</code> <code>challenge_1,</code> <code>challenge_2,</code> <code>challenge_3,</code> <code>challenge_4,</code> <code>challenge_5,</code> <code>challenge_6,</code> <code>challenge_7</code> <code>]</code>

B5.3.2.4 Footprint

ID	Value
state	<code>rec.attest_state</code>
challenge	<code>rec.attest_challenge</code>

B5.3.3 RSI_FEATURES command

Read feature register.

The following table indicates which feature register is returned depending on the index provided.

Index	Feature register
0	RSI feature register 0

See also:

- [Chapter A3 Feature discovery and configuration](#)

B5.3.3.1 Interface

B5.3.3.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000191
index	X1	63:0	UInt64	Feature register index

B5.3.3.1.2 Context

The RSI_FEATURES command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B5.3.3.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
value	X1	63:0	Bits64	Feature register value

B5.3.3.2 Failure conditions

The RSI_FEATURES command does not have any failure conditions.

B5.3.3.3 Success conditions

ID	Condition
value	<code>value == RsiFeatureRegisterEncode(realm, index)</code>

B5.3.3.4 Footprint

The RSI_FEATURES command does not have any footprint.

B5.3.4 RSI_HOST_CALL command

Make a Host call.

See also:

- [A4.5 Host call](#)
- [A5.2.4 RSI command access to a Protected IPA](#)

B5.3.4.1 Interface

B5.3.4.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000199
addr	X1	63:0	Address	IPA of the Host call data structure

B5.3.4.1.2 Context

The RSI_HOST_CALL command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rec	RmmRec	CurrentRec()	false	Current REC
data	RsiHostCall	RsiHostCallAt (addr)	false	Host call data structure
walk	RmmRttWalkResult	RttWalk (realm, addr, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result

B5.3.4.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.4.2 Failure conditions

ID	Condition
addr_align	pre: !AddrIsAligned (addr, 256) post: result == RSI_ERROR_INPUT
addr_bound	pre: !AddrIsProtected (addr, realm) post: result == RSI_ERROR_INPUT
addr_empty	pre: walk.rtte.ripas == EMPTY post: result == RSI_ERROR_INPUT

B5.3.4.2.1 Failure condition ordering

The RSI_HOST_CALL command does not have any failure condition orderings.

B5.3.4.3 Success conditions

The RSI_HOST_CALL command does not have any success conditions.

B5.3.4.4 Footprint

ID	Value
gprs	data.gprs

DRAFT

B5.3.5 RSI_IPA_STATE_GET command

Get RIPAS of a target IPA range.

See also:

- [A5.2 Realm view of memory management](#)
- [B5.3.6 RSI_IPA_STATE_SET command](#)

B5.3.5.1 Interface

B5.3.5.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000198
base	X1	63:0	Address	Base of target IPA region
top	X2	63:0	Address	End of target IPA region

B5.3.5.1.2 Context

The RSI_IPA_STATE_GET command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B5.3.5.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
out_top	X1	63:0	Address	Top of IPA region which has the reported RIPAS value
ripas	X2	7:0	RsiRipas	RIPAS value

The following unused bits of RSI_IPA_STATE_GET output values MBZ: X2[63:8].

If `result == RSI_SUCCESS` then all of the following are true:

- `out_top > base`
- `out_top <= top`
- All addresses within the range `[base, out_top)` have the RIPAS value `ripas`.

Note that the RIPAS of a Protected IPA can change at any time to DESTROYED without the Realm taking any action.

See also:

- [A5.2.6 Changes to RIPAS while Realm state is REALM_ACTIVE](#)

B5.3.5.2 Failure conditions

ID	Condition
base_align	pre: !AddrIsGranuleAligned(base) post: result == RSI_ERROR_INPUT
end_align	pre: !AddrIsGranuleAligned(top) post: result == RSI_ERROR_INPUT
size_valid	pre: UInt(top) <= UInt(base) post: result == RSI_ERROR_INPUT
rgn_bound	pre: !AddrRangeIsProtected(base, top, realm) post: result == RSI_ERROR_INPUT

B5.3.5.2.1 Failure condition ordering

The RSI_IPA_STATE_GET command does not have any failure condition orderings.

B5.3.5.3 Success conditions

The RSI_IPA_STATE_GET command does not have any success conditions.

B5.3.5.4 Footprint

The RSI_IPA_STATE_GET command does not have any footprint.

DRAFT

B5.3.6 RSI_IPA_STATE_SET command

Request RIPAS of a target IPA range to be changed to a specified value.

See also:

- [A5.2 Realm view of memory management](#)
- [A5.4 RIPAS change](#)
- [B5.3.5 RSI_IPA_STATE_GET command](#)

B5.3.6.1 Interface

B5.3.6.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000197
base	X1	63:0	Address	Base of target IPA region
top	X2	63:0	Address	Top of target IPA region
ripas	X3	7:0	RsiRipas	RIPAS value
flags	X4	63:0	RsiRipasChangeFlags	Flags

The following unused bits of RSI_IPA_STATE_SET input values SBZ: X3[63:8].

B5.3.6.1.2 Context

The RSI_IPA_STATE_SET command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rec	RmmRec	CurrentRec()	false	Current REC

B5.3.6.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
new_base	X1	63:0	Address	Base of IPA region which was not modified by the command
response	X2	0:0	RsiResponse	Whether the Host accepted or rejected the request

The following unused bits of RSI_IPA_STATE_SET output values MBZ: X2[63:1].

If the Host rejects the request then:

- `result == RSI_SUCCESS`
- `new_base == base`
- `response == RSI_REJECT`

B5.3.6.2 Failure conditions

ID	Condition
base_align	pre: !AddrIsGranuleAligned(base) post: result == RSI_ERROR_INPUT
top_align	pre: !AddrIsGranuleAligned(top) post: result == RSI_ERROR_INPUT
size_valid	pre: UInt(top) <= UInt(base) post: result == RSI_ERROR_INPUT
rgn_bound	pre: !AddrRangeIsProtected(base, top, realm) post: result == RSI_ERROR_INPUT
ripas_valid	pre: (ripas != RSI_EMPTY) && (ripas != RSI_RAM) post: result == RSI_ERROR_INPUT

B5.3.6.2.1 Failure condition ordering

The RSI_IPA_STATE_SET command does not have any failure condition orderings.

B5.3.6.3 Success conditions

ID	Condition
new_base	new_base == rec.ripas_addr
response	response == RecRipasResponseToRsi(rec)

B5.3.6.4 Footprint

The RSI_IPA_STATE_SET command does not have any footprint.

B5.3.7 RSI_MEASUREMENT_EXTEND command

Extend Realm Extensible Measurement (REM) value.

B5.3.7.1 Interface

B5.3.7.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000193
index	X1	63:0	UInt64	Measurement index
size	X2	63:0	UInt64	Measurement size in bytes
value_0	X3	63:0	Bits64	Doubleword 0 of the measurement value
value_1	X4	63:0	Bits64	Doubleword 1 of the measurement value
value_2	X5	63:0	Bits64	Doubleword 2 of the measurement value
value_3	X6	63:0	Bits64	Doubleword 3 of the measurement value
value_4	X7	63:0	Bits64	Doubleword 4 of the measurement value
value_5	X8	63:0	Bits64	Doubleword 5 of the measurement value
value_6	X9	63:0	Bits64	Doubleword 6 of the measurement value
value_7	X10	63:0	Bits64	Doubleword 7 of the measurement value

B5.3.7.1.2 Context

The RSI_MEASUREMENT_EXTEND command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
realm_pre	RmmRealm	CurrentRealm()	true	Current Realm
meas_pre	RmmRealmMeasurement	realm_pre.measurements[index]	true	Previous measurement value

B5.3.7.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.7.2 Failure conditions

ID	Condition
index_bound	pre: index < 1 index > 4 post: result == RSI_ERROR_INPUT
size_bound	pre: size > 64 post: result == RSI_ERROR_INPUT

B5.3.7.2.1 Failure condition ordering

The RSI_MEASUREMENT_EXTEND command does not have any failure condition orderings.

B5.3.7.3 Success conditions

ID	Condition
realm_meas	realm.measurements[index] == RemExtend(realm.hash_algo, meas_pre, [value_0, value_1, value_2, value_3, value_4, value_5, value_6, value_7][(RMM_REALM_MEASUREMENT_WIDTH-1):0], size * 8)

B5.3.7.4 Footprint

ID	Value
realm_meas	realm.measurements[index]

B5.3.8 RSI_MEASUREMENT_READ command

Read measurement for the current Realm.

See also:

- [A7.1 Realm measurements](#)
- [D1.2.1 Realm creation flow](#)

B5.3.8.1 Interface

B5.3.8.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000192
index	X1	63:0	UInt64	Measurement index

`index` 0 selects the RIM. An `index` of 1 or greater selects the corresponding REM.

B5.3.8.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
value_0	X1	63:0	Bits64	Doubleword 0 of the Realm measurement identified by “index”
value_1	X2	63:0	Bits64	Doubleword 1 of the Realm measurement identified by “index”
value_2	X3	63:0	Bits64	Doubleword 2 of the Realm measurement identified by “index”
value_3	X4	63:0	Bits64	Doubleword 3 of the Realm measurement identified by “index”
value_4	X5	63:0	Bits64	Doubleword 4 of the Realm measurement identified by “index”
value_5	X6	63:0	Bits64	Doubleword 5 of the Realm measurement identified by “index”
value_6	X7	63:0	Bits64	Doubleword 6 of the Realm measurement identified by “index”
value_7	X8	63:0	Bits64	Doubleword 7 of the Realm measurement identified by “index”

If the size of the measurement value is smaller than 512 bits, the output values are padded with zeroes.

B5.3.8.2 Failure conditions

ID	Condition
index_bound	pre: index > 4 post: result == RSI_ERROR_INPUT

B5.3.8.3 Success conditions

The RSI_MEASUREMENT_READ command does not have any success conditions.

B5.3.8.4 Footprint

The RSI_MEASUREMENT_READ command does not have any footprint.

DRAFT

B5.3.9 RSI_MEM_GET_PERM_VALUE command

Get overlay permission value for a specified (plane index, overlay permission index) tuple.

See also:

- [A10.3.2 Stage 2 Access Permissions within a multi-Plane Realm](#)

B5.3.9.1 Interface

B5.3.9.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A0
plane_index	X1	63:0	UInt64	Plane index
perm_index	X2	63:0	UInt64	Permission index

B5.3.9.1.2 Context

The RSI_MEM_GET_PERM_VALUE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B5.3.9.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
value	X1	63:0	Bits64	Memory permission value

B5.3.9.2 Failure conditions

ID	Condition
plane_bound	pre: plane_index > realm.num_aux_planes post: result == RSI_ERROR_INPUT
perm_bound	pre: perm_index >= RMM_NUM_PERM_OVERLAY_INDICES post: result == RSI_ERROR_INPUT

B5.3.9.2.1 Failure condition ordering

The RSI_MEM_GET_PERM_VALUE command does not have any failure condition orderings.

B5.3.9.3 Success conditions

ID	Condition
label	<code>value == realm.overlay_perms[plane_index].values[perm_index]</code>

B5.3.9.4 Footprint

The RSI_MEM_GET_PERM_VALUE command does not have any footprint.

DRAFT

B5.3.10 RSI_MEM_SET_PERM_INDEX command

Set overlay permission index for a specified IPA range.

See also:

- [A10.3.2 Stage 2 Access Permissions within a multi-Plane Realm](#)

B5.3.10.1 Interface

B5.3.10.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A1
base	X1	63:0	Address	Base of target IPA region
top	X2	63:0	Address	Top of target IPA region
perm_index	X3	63:0	UInt64	Permission index
cookie	X4	63:0	Bits64	Cookie value

B5.3.10.1.2 Context

The RSI_MEM_SET_PERM_INDEX command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rec	RmmRec	CurrentRec()	false	Current REC

B5.3.10.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
new_base	X1	63:0	Address	Base of IPA region which was not modified by the command
response	X2	0:0	RsiResponse	Whether the Host accepted or rejected the request
new_cookie	X3	63:0	Bits64	New cookie value

The following unused bits of RSI_MEM_SET_PERM_INDEX output values MBZ: X2[63:1].

B5.3.10.2 Failure conditions

ID	Condition
base_align	pre: <code>!AddrIsGranuleAligned(base)</code> post: <code>result == RSI_ERROR_INPUT</code>

ID	Condition
top_align	pre: !AddrIsGranuleAligned(top) post: result == RSI_ERROR_INPUT
size_valid	pre: UInt(top) <= UInt(base) post: result == RSI_ERROR_INPUT
rgn_bound	pre: !AddrRangeIsProtected(base, top, realm) post: result == RSI_ERROR_INPUT
perm_bound	pre: perm_index >= RMM_NUM_PERM_OVERLAY_INDICES post: result == RSI_ERROR_INPUT
cookie	pre: Cookie is invalid post: result == RSI_ERROR_INPUT

B5.3.10.2.1 Failure condition ordering

The RSI_MEM_SET_PERM_INDEX command does not have any failure condition orderings.

B5.3.10.3 Success conditions

ID	Condition
locked	realm.overlay_locked[perm_index] == MEM_PERM_LOCKED
new_base	new_base == rec.s2ap_addr
response	response == RecS2APResponseToRsi(rec)

B5.3.10.4 Footprint

The RSI_MEM_SET_PERM_INDEX command does not have any footprint.

B5.3.11 RSI_MEM_SET_PERM_VALUE command

Set overlay permission value for a specified (plane index, overlay permission index) tuple.

See also:

- [A10.3.2 Stage 2 Access Permissions within a multi-Plane Realm](#)

B5.3.11.1 Interface

B5.3.11.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A2
plane_index	X1	63:0	UInt64	Plane index
perm_index	X2	63:0	UInt64	Permission index
value	X3	63:0	Bits64	Memory permission value

B5.3.11.1.2 Context

The RSI_MEM_SET_PERM_VALUE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B5.3.11.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.11.2 Failure conditions

ID	Condition
plane_bound	pre: (plane_index == 0 plane_index > realm.num_aux_planes) post: result == RSI_ERROR_INPUT
perm_bound	pre: perm_index >= RMM_NUM_PERM_OVERLAY_INDICES post: result == RSI_ERROR_INPUT
locked	pre: realm.overlay_locked[perm_index] == MEM_PERM_LOCKED post: result == RSI_ERROR_INPUT
supported	pre: ! MemPermLabelSupported (value) post: result == RSI_ERROR_INPUT

B5.3.11.2.1 Failure condition ordering

The RSI_MEM_SET_PERM_VALUE command does not have any failure condition orderings.

B5.3.11.3 Success conditions

ID	Condition
label	<code>realm.overlay_perms[plane_index].values[perm_index] == value</code>

B5.3.11.4 Footprint

The RSI_MEM_SET_PERM_VALUE command does not have any footprint.

DRAFT

B5.3.12 RSI_PLANE_ENTER command

Enter a Plane.

See also:

- [A5.2.4 RSI command access to a Protected IPA](#)
- [A10.2 Planes exception model](#)

B5.3.12.1 Interface

B5.3.12.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A3
plane_idx	X1	63:0	UInt64	Index of target Plane
run_ptr	X2	63:0	Address	IPA of PlaneRun object

B5.3.12.1.2 Context

The RSI_PLANE_ENTER command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
run	RsiPlaneRun	RsiPlaneRunAt (realm, run_ptr)	false	PlaneRun object
walk	RmmRttWalkResult	RttWalk (realm, run_ptr, RMM_RTT_PAGE_LEVEL , RMM_RTT_TREE_PRIMARY)	false	RTT walk result

B5.3.12.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.12.2 Failure conditions

ID	Condition
idx_bound	pre: (plane_idx == 0 plane_idx > realm.num_aux_planes) post: result == RSI_ERROR_INPUT
run_align	pre: !AddrIsGranuleAligned (run_ptr) post: result == RSI_ERROR_INPUT
run_bound	pre: !AddrIsProtected (run_ptr, realm) post: result == RSI_ERROR_INPUT

ID	Condition
run_empty	pre: walk.rtte.ripas == EMPTY post: result == RSI_ERROR_INPUT

B5.3.12.2.1 Failure condition ordering

The RSI_PLANE_ENTER command does not have any failure condition orderings.

B5.3.12.3 Success conditions

ID	Condition
plane_exit	run.exit contains Plane exit syndrome information.

B5.3.12.4 Footprint

The RSI_PLANE_ENTER command does not have any footprint.

DRAFT

B5.3.13 RSI_PLANE_REG_READ command

Read a Plane register.

See also:

- [A10.2.6 Pn system registers](#)

B5.3.13.1 Interface

B5.3.13.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001AE
plane_idx	X1	63:0	UInt64	Index of target Plane
encoding	X2	63:0	Bits64	Encoding of target register

The encoding value is an architecturally-defined system register encoding.

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)

B5.3.13.1.2 Context

The RSI_PLANE_REG_READ command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B5.3.13.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
value	X1	63:0	Bits64	Value of target register

B5.3.13.2 Failure conditions

ID	Condition
idx_bound	pre: plane_idx > realm.num_aux_planes post: result == RSI_ERROR_INPUT
reg_valid	pre: !PlaneRegIsValid (realm, encoding) post: result == RSI_ERROR_INPUT

B5.3.13.2.1 Failure condition ordering

The RSI_PLANE_REG_READ command does not have any failure condition orderings.

B5.3.13.3 Success conditions

ID	Condition
value	value == <code>PlaneRegValue</code> (realm, plane_idx, encoding)

B5.3.13.4 Footprint

The RSI_PLANE_REG_READ command does not have any footprint.

DRAFT

B5.3.14 RSI_PLANE_REG_WRITE command

Write a Plane register.

See also:

- [A10.2.6 Pn system registers](#)

B5.3.14.1 Interface

B5.3.14.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001AF
plane_idx	X1	63:0	UInt64	Index of target Plane
encoding	X2	63:0	Bits64	Encoding of target register
value	X3	63:0	Bits64	Value to write to target register

The encoding value is an architecturally-defined system register encoding.

See also:

- [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)

B5.3.14.1.2 Context

The RSI_PLANE_REG_WRITE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B5.3.14.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.14.2 Failure conditions

ID	Condition
idx_bound	pre: plane_idx > realm.num_aux_planes post: result == RSI_ERROR_INPUT
reg_valid	pre: ! PlaneRegIsValid (realm, encoding) post: result == RSI_ERROR_INPUT

B5.3.14.2.1 Failure condition ordering

The RSI_PLANE_REG_WRITE command does not have any failure condition orderings.

B5.3.14.3 Success conditions

ID	Condition
value	<code>PlaneRegValue(realm, plane_idx, encoding) == value</code>

B5.3.14.4 Footprint

The RSI_PLANE_REG_WRITE command does not have any footprint.

DRAFT

B5.3.15 RSI_RDEV_CONTINUE command

Continue an interruptible Realm device operation.

See also:

- [A9.5.1 Interruptible Realm device operations](#)

B5.3.15.1 Interface

B5.3.15.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A4
vdev_id	X1	63:0	Bits64	Realm device identifier
inst_id	X2	63:0	UInt64	Device instance identifier

B5.3.15.1.2 Context

The RSI_RDEV_CONTINUE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromInstId() realm, inst_id)	false	Realm device

B5.3.15.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

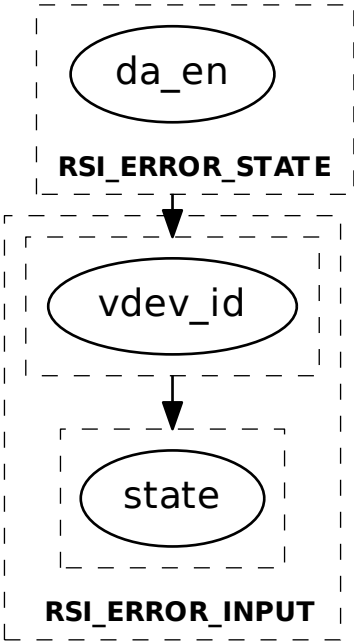
B5.3.15.2 Failure conditions

ID	Condition
da_en	pre: realm.feat_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: ! RdevIdsAreValid (realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT
state	pre: (rdev.state != RDEV_UNLOCKED_BUSY && rdev.state != RDEV_LOCKED_BUSY && rdev.state != RDEV_STARTED_BUSY && rdev.state != RDEV_STOPPING) post: result == RSI_ERROR_INPUT

B5.3.15.2.1 Failure condition ordering

[da_en] < [vdev_id]
[vdev_id] < [state]

DRAFT



B5.3.15.3 Success conditions

ID	Condition
error	pre: (DeviceCommunicate(rdev) == DEV_COMM_ERROR && rdev.state != RDEV_STOPPING) post: rdev.state == RDEV_ERROR
new	pre: (DeviceCommunicate(rdev) == DEV_COMM_IDLE && rdev.state == RDEV_UNLOCKED_BUSY && rdev.operation != RDEV_OP_LOCK) post: rdev.state == RDEV_UNLOCKED
to_locked	pre: (DeviceCommunicate(rdev) == DEV_COMM_IDLE && rdev.state == RDEV_UNLOCKED_BUSY && rdev.operation == RDEV_OP_LOCK) post: rdev.state == RDEV_LOCKED
locked	pre: (DeviceCommunicate(rdev) == DEV_COMM_IDLE && rdev.state == RDEV_LOCKED_BUSY && rdev.operation != RDEV_OP_LOCK) post: rdev.state == RDEV_LOCKED

ID	Condition
to_started	<pre>pre: (DeviceCommunicate(rdev) == DEV_COMM_IDLE && rdev.state == RDEV_LOCKED_BUSY && rdev.operation == RDEV_OP_START) post: rdev.state == RDEV_STARTED</pre>
started	<pre>pre: (DeviceCommunicate(rdev) == DEV_COMM_IDLE && rdev.state == RDEV_STARTED_BUSY) post: rdev.state == RDEV_STARTED</pre>
stopped	<pre>pre: (DeviceCommunicate(rdev) != DEV_COMM_ACTIVE && rdev.state == RDEV_STOPPING) post: rdev.state == RDEV_STOPPED</pre>

B5.3.15.4 Footprint

ID	Value
state	rdev.state

DRAFT

B5.3.16 RSI_RDEV_GET_INFO command

Get information for a device.

Device configuration information, including digests of attestation evidence for the device are written to an RsiDeviceInfo structure, at an address specified by the caller. Digests are calculated using the PDEV Hash Algorithm.

See also:

- [A5.2.4 RSI command access to a Protected IPA](#)
- [A9.5.2 Realm retrieval of device attestation evidence](#)
- [B5.3.25 RSI_REALM_CONFIG command](#)
- [B5.4.4 RsiDeviceInfo type](#)

B5.3.16.1 Interface

B5.3.16.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A5
vdev_id	X1	63:0	Bits64	Realm device identifier
inst_id	X2	63:0	UInt64	Device instance identifier
addr	X3	63:0	Address	IPA of the Granule to which the configuration data will be written

B5.3.16.1.2 Context

The RSI_RDEV_GET_INFO command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromInstId(realm, inst_id)	false	Realm device
vdev	RmmVdev	VdevAt(rdev.vdev_ptr)	false	Virtual device
pdev	RmmPdev	PdevAt(vdev.pdev)	false	Physical device
cfg	RsiDeviceInfo	RsiDeviceInfoAt(addr)	false	Device configuration
walk	RmmRttWalkResult	RttWalk(realm, addr, RMM_RTT_PAGE_LEVEL, RMM_RTT_TREE_PRIMARY)	false	RTT walk result

B5.3.16.1.3 Output values

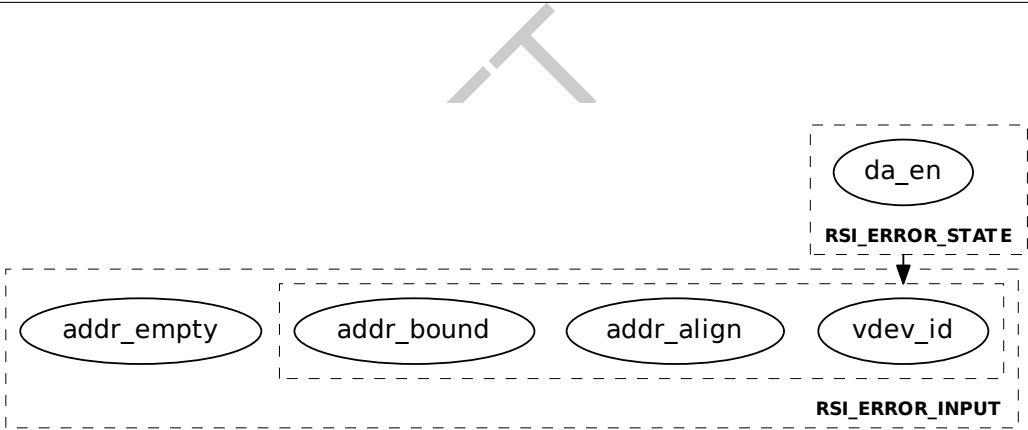
Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.16.2 Failure conditions

ID	Condition
da_en	pre: realm.feat_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: !RdevIdsAreValid(realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT
addr_align	pre: !AddrIsGranuleAligned(addr) post: result == RSI_ERROR_INPUT
addr_bound	pre: !AddrIsProtected(addr, realm) post: result == RSI_ERROR_INPUT
addr_empty	pre: walk.rtte.ripas == EMPTY post: result == RSI_ERROR_INPUT

B5.3.16.2.1 Failure condition ordering

[da_en] < [vdev_id, addr_align, addr_bound]



B5.3.16.3 Success conditions

ID	Condition
hash_algo	<code>Equal</code> (cfg.hash_algo, pdev.hash_algo)

B5.3.16.4 Footprint

The RSI_RDEV_GET_INFO command does not have any footprint.

B5.3.17 RSI_RDEV_GET_INSTANCE_ID command

Get instance ID for a device.

See also:

- [A9.2.2 Mapping from virtual device ID to VDEV object](#)

B5.3.17.1 Interface

B5.3.17.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400019C
vdev_id	X1	63:0	Bits64	Realm device identifier

B5.3.17.1.2 Context

The RSI_RDEV_GET_INSTANCE_ID command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromVdevId(realm, vdev_id)	false	Realm device
vdev	RmmVdev	VdevAt(rdev.vdev_ptr)	false	Virtual device

B5.3.17.1.3 Output values

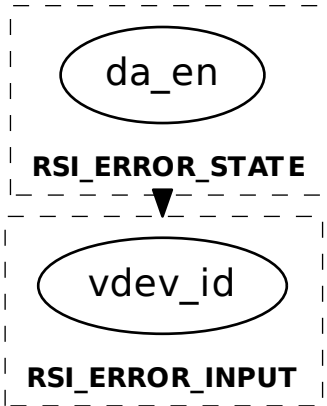
Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
inst_id	X1	63:0	UInt64	Instance ID

B5.3.17.2 Failure conditions

ID	Condition
da_en	pre: realm.feats_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: !RdevIdIsValid(realm, vdev_id) post: result == RSI_ERROR_INPUT

B5.3.17.2.1 Failure condition ordering

[da_en] < [vdev_id]



B5.3.17.3 Success conditions

ID	Condition
inst_id	inst_id == vdev.inst_id

B5.3.17.4 Footprint

The RSI_RDEV_GET_INSTANCE_ID command does not have any footprint.

B5.3.18 RSI_RDEV_GET_INTERFACE_REPORT command

Get Realm device interface report.

See also:

- [A9.5.2 Realm retrieval of device attestation evidence](#)

B5.3.18.1 Interface

B5.3.18.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A6
vdev_id	X1	63:0	Bits64	Realm device identifier
inst_id	X2	63:0	UInt64	Device instance identifier
version_max	X3	63:0	UInt64	Maximum TDISP version accepted by caller

B5.3.18.1.2 Context

The RSI_RDEV_GET_INTERFACE_REPORT command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromInstId(realm, inst_id)	false	Realm device

B5.3.18.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
version	X1	63:0	UInt64	TDISP version

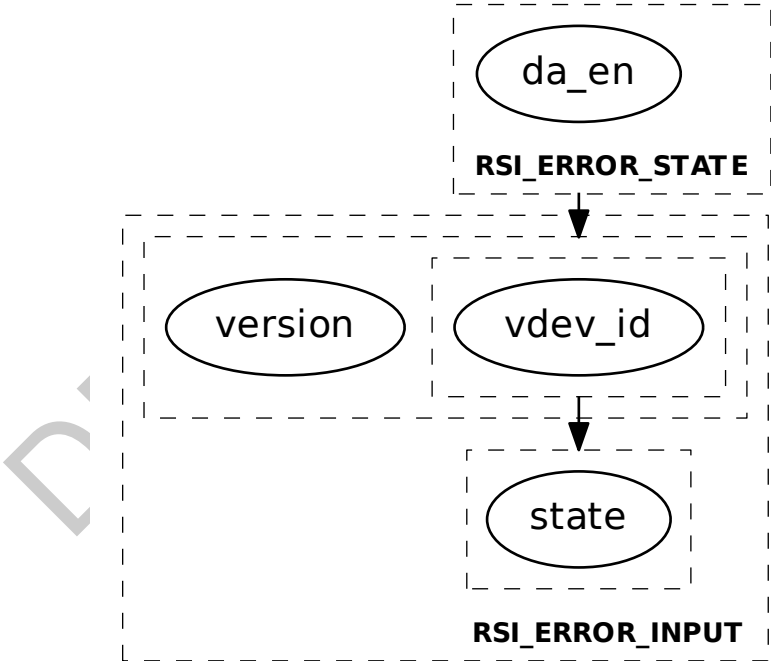
B5.3.18.2 Failure conditions

ID	Condition
da_en	pre: realm.feats_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: !RdevIdsAreValid(realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT
version	pre: TDISP version is not supported. post: result == RSI_ERROR_INPUT

ID	Condition
state	<pre>pre: (rdev.state != RDEV_LOCKED && rdev.state != RDEV_STARTED) post: result == RSI_ERROR_INPUT</pre>

B5.3.18.2.1 Failure condition ordering

<pre>[da_en] < [vdev_id, version] [vdev_id] < [state]</pre>



B5.3.18.3 Success conditions

ID	Condition
locked	<pre>pre: rdev.state == RDEV_LOCKED post: rdev.state == RDEV_LOCKED_BUSY</pre>
started	<pre>pre: rdev.state == RDEV_STARTED post: rdev.state == RDEV_STARTED_BUSY</pre>
operation	<pre>rdev.operation == RDEV_OP_GET_INTERFACE_REPORT</pre>

B5.3.18.4 Footprint

ID	Value
state	rdev.state
operation	rdev.operation

DRAFT

B5.3.19 RSI_RDEV_GET_MEASUREMENTS command

Get Realm device measurements.

See also:

- [A5.2.4 RSI command access to a Protected IPA](#)
- [A9.5.2 Realm retrieval of device attestation evidence](#)

B5.3.19.1 Interface

B5.3.19.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A7
vdev_id	X1	63:0	Bits64	Virtual device identifier
inst_id	X2	63:0	UInt64	Device instance identifier
op	X3	63:0	Bits64	Measurement operation as defined as Param2 in SPDMM GET_MEASUREMENTS request
flags	X4	63:0	Bits64	If set to 0x1, request RawBitStream. Otherwise, request hash of the measurement as defined by request attributes in SPDMM GET_MEASUREMENTS command.

B5.3.19.1.2 Context

The RSI_RDEV_GET_MEASUREMENTS command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromInstId(realm, inst_id)	false	Realm device

B5.3.19.1.3 Output values

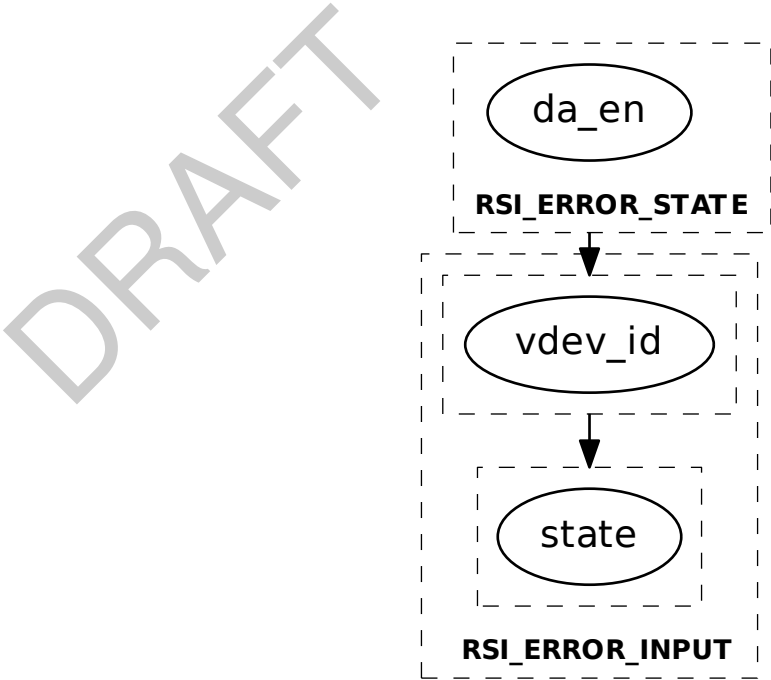
Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.19.2 Failure conditions

ID	Condition
da_en	pre: realm.feat_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: !RdevIdsAreValid(realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT
state	pre: (rdev.state != RDEV_UNLOCKED && rdev.state != RDEV_LOCKED && rdev.state != RDEV_STARTED) post: result == RSI_ERROR_INPUT

B5.3.19.2.1 Failure condition ordering

[da_en] < [vdev_id]
[vdev_id] < [state]



B5.3.19.3 Success conditions

ID	Condition
new	pre: rdev.state == RDEV_UNLOCKED post: rdev.state == RDEV_UNLOCKED_BUSY
locked	pre: rdev.state == RDEV_LOCKED post: rdev.state == RDEV_LOCKED_BUSY

ID	Condition
started	pre: rdev.state == RDEV_STARTED post: rdev.state == RDEV_STARTED_BUSY
operation	rdev.operation == RDEV_OP_GET_MEASUREMENTS

B5.3.19.4 Footprint

ID	Value
state	rdev.state
operation	rdev.operation

DRAFT

B5.3.20 RSI_RDEV_GET_STATE command

Get state of a Realm device.

See also:

- [A9.5.5 Realm device lifecycle](#)

B5.3.20.1 Interface

B5.3.20.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A8
vdev_id	X1	63:0	Bits64	Realm device identifier
inst_id	X2	63:0	UInt64	Device instance identifier

B5.3.20.1.2 Context

The RSI_RDEV_GET_STATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromInstId(realm, inst_id)	false	Realm device

B5.3.20.1.3 Output values

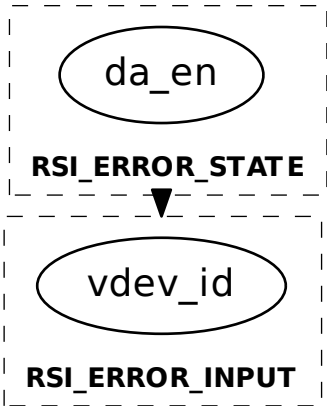
Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
state	X1	63:0	RsiDeviceState	Realm device state

B5.3.20.2 Failure conditions

ID	Condition
da_en	pre: realm.feat_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: !RdevIdsAreValid (realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT

B5.3.20.2.1 Failure condition ordering

[da_en] < [vdev_id]



B5.3.20.3 Success conditions

ID	Condition
state	<code>Equal(state, rdev.state)</code>

B5.3.20.4 Footprint

The RSI_RDEV_GET_STATE command does not have any footprint.

B5.3.21 RSI_RDEV_LOCK command

Lock a Realm device.

See also:

- [A9.5.5 Realm device lifecycle](#)

B5.3.21.1 Interface

B5.3.21.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001A9
vdev_id	X1	63:0	Bits64	Realm device identifier
inst_id	X2	63:0	UInt64	Device instance identifier

B5.3.21.1.2 Context

The RSI_RDEV_LOCK command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromInstId (realm, inst_id)	false	Realm device

B5.3.21.1.3 Output values

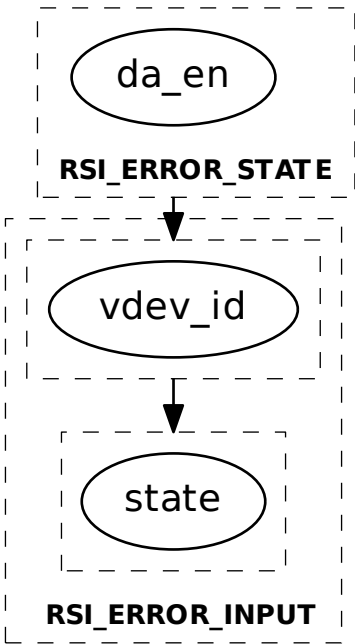
Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.21.2 Failure conditions

ID	Condition
da_en	pre: realm.featur_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: ! RdevIdsAreValid (realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT
state	pre: rdev.state != RDEV_UNLOCKED post: result == RSI_ERROR_INPUT

B5.3.21.2.1 Failure condition ordering

[da_en] < [vdev_id]
[vdev_id] < [state]



B5.3.21.3 Success conditions

ID	Condition
state	rdev.state == RDEV_UNLOCKED_BUSY
operation	rdev.operation == RDEV_OP_LOCK

B5.3.21.4 Footprint

ID	Value
state	rdev.state
operation	rdev.operation

B5.3.22 RSI_RDEV_START command

Start a Realm device.

See also:

- [A9.5.5 Realm device lifecycle](#)

B5.3.22.1 Interface

B5.3.22.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001AA
vdev_id	X1	63:0	Bits64	Realm device identifier
inst_id	X2	63:0	UInt64	Device instance identifier

B5.3.22.1.2 Context

The RSI_RDEV_START command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromInstId (realm, inst_id)	false	Realm device

B5.3.22.1.3 Output values

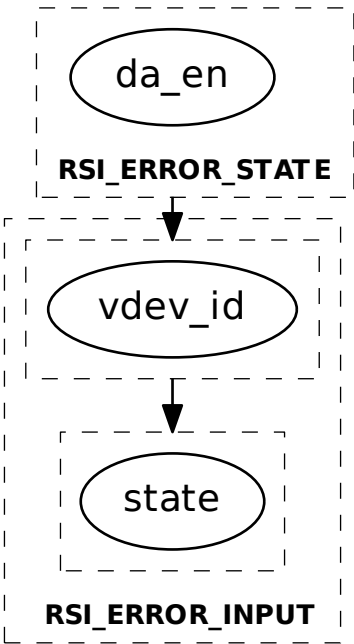
Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.22.2 Failure conditions

ID	Condition
da_en	pre: realm.feats_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: ! RdevIdsAreValid (realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT
state	pre: rdev.state != RDEV_LOCKED post: result == RSI_ERROR_INPUT

B5.3.22.2.1 Failure condition ordering

[da_en] < [vdev_id]
[vdev_id] < [state]



B5.3.22.3 Success conditions

ID	Condition
state	rdev.state == RDEV_LOCKED_BUSY
operation	rdev.operation == RDEV_OP_START

B5.3.22.4 Footprint

ID	Value
state	rdev.state
operation	rdev.operation

B5.3.23 RSI_RDEV_STOP command

Stop a Realm device.

See also:

- [A9.5.5 Realm device lifecycle](#)

B5.3.23.1 Interface

B5.3.23.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001AB
vdev_id	X1	63:0	Bits64	Realm device identifier
inst_id	X2	63:0	UInt64	Device instance identifier

B5.3.23.1.2 Context

The RSI_RDEV_STOP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rdev	RmmRdev	RdevFromInstId (realm, inst_id)	false	Realm device

B5.3.23.1.3 Output values

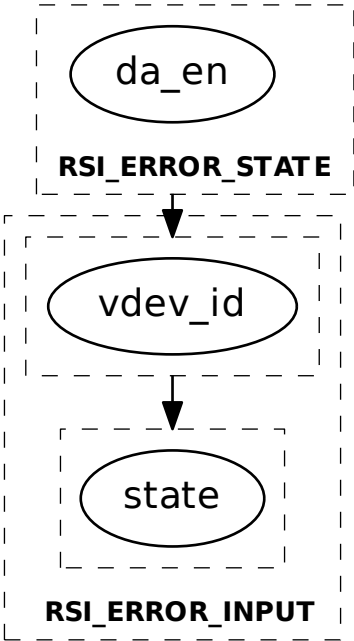
Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.23.2 Failure conditions

ID	Condition
da_en	pre: realm.feat_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: ! RdevIdsAreValid (realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT
state	pre: (rdev.state != RDEV_UNLOCKED && rdev.state != RDEV_LOCKED && rdev.state != RDEV_STARTED && rdev.state != RDEV_ERROR) post: result == RSI_ERROR_INPUT

B5.3.23.2.1 Failure condition ordering

[da_en] < [vdev_id]
[vdev_id] < [state]



B5.3.23.3 Success conditions

ID	Condition
state	rdev.state == RDEV_STOPPING

B5.3.23.4 Footprint

ID	Value
state	rdev.state

B5.3.24 RSI_RDEV_VALIDATE_MAPPING command

Validate Realm device memory mappings.

See also:

- [A9.5.3 Realm validation of device memory mappings](#)

B5.3.24.1 Interface

B5.3.24.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC40001AC
vdev_id	X1	63:0	Bits64	Realm device identifier
inst_id	X2	63:0	UInt64	Device instance identifier
ipa_base	X3	63:0	Address	Base of target IPA region
ipa_top	X4	63:0	Address	Top of target IPA region
pa_base	X5	63:0	Address	Base of target PA region
flags	X6	63:0	RsiDevMemFlags	Flags

B5.3.24.1.2 Context

The RSI_RDEV_VALIDATE_MAPPING command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
rec	RmmRec	CurrentRec()	false	Current REC
rdev	RmmRdev	RdevFromInstId() realm, inst_id)	false	Realm device

B5.3.24.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
new_ipa_base	X1	63:0	Address	Base of IPA region which was not modified by the command
response	X2	0:0	RsiResponse	Whether the Host accepted or rejected the request

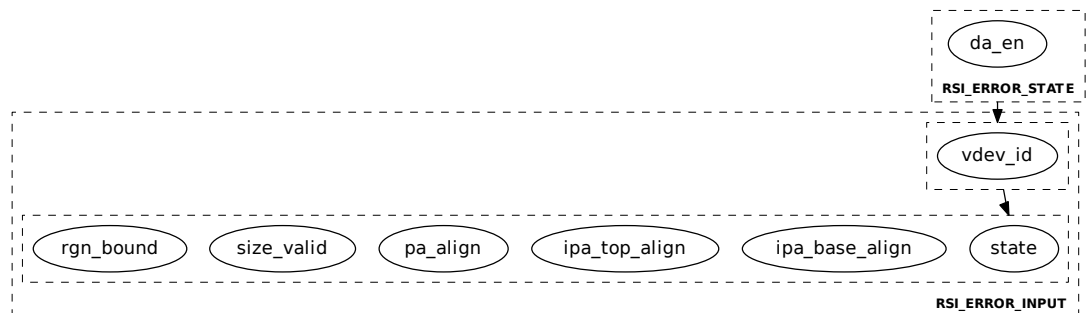
The following unused bits of RSI_RDEV_VALIDATE_MAPPING output values MBZ: X2[63:1].

B5.3.24.2 Failure conditions

ID	Condition
da_en	pre: realm.feat_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: !RdevIdsAreValid(realm, vdev_id, inst_id) post: result == RSI_ERROR_INPUT
state	pre: (rdev.state != RDEV_LOCKED && rdev.state != RDEV_STARTED) post: result == RSI_ERROR_INPUT
ipa_base_align	pre: !AddrIsGranuleAligned(ipa_base) post: result == RSI_ERROR_INPUT
ipa_top_align	pre: !AddrIsGranuleAligned(ipa_top) post: result == RSI_ERROR_INPUT
pa_align	pre: !AddrIsGranuleAligned(pa_base) post: result == RSI_ERROR_INPUT
size_valid	pre: UInt(ipa_top) <= UInt(ipa_base) post: result == RSI_ERROR_INPUT
rgn_bound	pre: !AddrRangeIsProtected(ipa_base, ipa_top, realm) post: result == RSI_ERROR_INPUT

B5.3.24.2.1 Failure condition ordering

```
[da_en] < [vdev_id]
[vdev_id] < [state, ipa_base_align, ipa_top_align, pa_align,
size_valid, rgn_bound]
```



B5.3.24.3 Success conditions

ID	Condition
new_ipa_base	new_ipa_base == rec.dev_mem_addr
response	response == RecDevMemResponseToRsi(rec)

B5.3.24.4 Footprint

The RSI_RDEV_VALIDATE_MAPPING command does not have any footprint.

B5.3.25 RSI_REALM_CONFIG command

Read configuration for the current Realm.

See also:

- [A5.2.4 RSI command access to a Protected IPA](#)

B5.3.25.1 Interface

B5.3.25.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000196
addr	X1	63:0	Address	IPA of the Granule to which the configuration data will be written

B5.3.25.1.2 Context

The RSI_REALM_CONFIG command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
cfg	RsiRealmConfig	RsiRealmConfigAt(addr)	false	Realm configuration
walk	RmmRttWalkResult	RttWalk(realm, addr, RMM_RTT_PAGE_LEVEL, RMM_RTT_TREE_PRIMARY)	false	RTT walk result

B5.3.25.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status

B5.3.25.2 Failure conditions

ID	Condition
addr_align	pre: !AddrIsGranuleAligned(addr) post: result == RSI_ERROR_INPUT
addr_bound	pre: !AddrIsProtected(addr, realm) post: result == RSI_ERROR_INPUT
addr_empty	pre: walk.rtte.ripas == EMPTY post: result == RSI_ERROR_INPUT

B5.3.25.2.1 Failure condition ordering

The RSI_REALM_CONFIG command does not have any failure condition orderings.

B5.3.25.3 Success conditions

ID	Condition
ipa_width	<code>cfg.ipa_width == realm.ipa_width</code>
hash_algo	<code>Equal(cfg.hash_algo, realm.hash_algo)</code>
num_aux_planes	<code>cfg.num_aux_planes == realm.num_aux_planes</code>

B5.3.25.4 Footprint

The RSI_REALM_CONFIG command does not have any footprint.

DRAFT

B5.3.26 RSI_VERSION command

Returns RSI version.

On calling this command, the Realm provides a requested RSI version.

The output values include a status code and two revisions which are supported by the RMM: a *lower revision* and a *higher revision*.

- The *higher revision* value is the highest interface revision which is supported by the RMM.
- The *lower revision* is less than or equal to the *higher revision*.

The status code and *lower revision* output values indicate which of the following is true, in order of precedence:

- a) The RMM supports an interface revision which is compatible with the requested revision.
 - The status code is RSI_SUCCESS.
 - The *lower revision* is equal to the requested revision.
- b) The RMM does not support an interface revision which is compatible with the requested revision. The RMM supports an interface revision which is incompatible with and less than the requested revision.
 - The status code is RSI_ERROR_INPUT.
 - The *lower revision* is the highest interface revision which is both less than the requested revision and supported by the RMM.
- c) The RMM does not support an interface revision which is compatible with the requested revision. The RMM supports an interface revision which is incompatible with and greater than the requested revision.
 - The status code is RSI_ERROR_INPUT.
 - The *lower revision* is equal to the *higher revision*.

See also:

- [Chapter B2 Interface versioning](#)
- [B5.1 RSI version](#)

B5.3.26.1 Interface

B5.3.26.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000190
req	X1	63:0	RsiInterfaceVersion	Requested interface revision

B5.3.26.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
lower	X1	63:0	RsiInterfaceVersion	Lower supported interface revision
higher	X2	63:0	RsiInterfaceVersion	Higher supported interface revision

B5.3.26.2 Failure conditions

ID	Condition
incompat_lower	<pre>pre: (!RsiVersionIsSupported(req) && RsiVersionLowerIsSupported(req)) post: (result == RSI_ERROR_INPUT && VersionEqual(lower, RsiVersionHighestBelow(req)) && VersionEqual(higher, RsiVersionHighest()))</pre>
incompat_higher	<pre>pre: (!RsiVersionIsSupported(req) && !RsiVersionLowerIsSupported(req) && RsiVersionHigherIsSupported(req)) post: (result == RSI_ERROR_INPUT && VersionEqual(lower, higher) && VersionEqual(higher, RsiVersionHighest()))</pre>

B5.3.26.2.1 Failure condition ordering

The RSI_VERSION command does not have any failure condition orderings.

B5.3.26.3 Success conditions

ID	Condition
lower	<code>VersionEqual(lower, req)</code>
higher	<code>VersionEqual(higher, RsiVersionHighest())</code>

B5.3.26.4 Footprint

The RSI_VERSION command does not have any footprint.

B5.3.27 RSI_VSMMU_ACTIVATE command

Activate a VSMMU.

See also:

- [A9.6.4 VSMMU validation](#)

B5.3.27.1 Interface

B5.3.27.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400019B
base	X1	63:0	Address	Base of target IPA region
top	X2	63:0	Address	Top of target IPA region

B5.3.27.1.2 Context

The RSI_VSMMU_ACTIVATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B5.3.27.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
new_base	X1	63:0	Address	Base of IPA region which was not modified by the command

B5.3.27.2 Failure conditions

ID	Condition
base_align	pre: !AddrIsGranuleAligned (base) post: result == RSI_ERROR_INPUT
top_align	pre: !AddrIsGranuleAligned (top) post: result == RSI_ERROR_INPUT
size_valid	pre: UInt (top) <= UInt (base) post: result == RSI_ERROR_INPUT
rgn_bound	pre: !AddrRangeIsProtected (base, top, realm) post: result == RSI_ERROR_INPUT

B5.3.27.2.1 Failure condition ordering

The RSI_VSMMU_ACTIVATE command does not have any failure condition orderings.

B5.3.27.3 Success conditions

The RSI_VSMMU_ACTIVATE command does not have any success conditions.

B5.3.27.4 Footprint

The RSI_VSMMU_ACTIVATE command does not have any footprint.

DRAFT

B5.3.28 RSI_VSMMU_GET_INFO command

Get information of a VSMMU.

See also:

- [A9.6.4 VSMMU validation](#)

B5.3.28.1 Interface

B5.3.28.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC400019A
addr	X1	63:0	Address	Base IPA of the VSMMU

B5.3.28.1.2 Context

The RSI_VSMMU_GET_INFO command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
walk	RmmRttWalkResult	RttWalk(realm, addr, RMM_RTT_PAGE_LEVEL, RMM_RTT_TREE_PRIMARY)	false	RTT walk result

B5.3.28.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command return status
top	X1	63:0	Address	Top IPA of the VSMMU

B5.3.28.2 Failure conditions

ID	Condition
addr_align	pre: !AddrIsGranuleAligned(addr) post: result == RSI_ERROR_INPUT
addr_bound	pre: !AddrIsProtected(addr, realm) post: result == RSI_ERROR_INPUT
rtte_state	pre: walk.rtte.state != ASSIGNED_VSMMU post: result == RSI_ERROR_INPUT
vsmmu_base	pre: addr != VsmmuAt(walk.rtte.addr).reg_base post: result == RSI_ERROR_INPUT

B5.3.28.2.1 Failure condition ordering

The RSI_VSMMU_GET_INFO command does not have any failure condition orderings.

B5.3.28.3 Success conditions

ID	Condition
vsmmu_base	top == VsmmuAt (walk.rtte.addr).reg_top

B5.3.28.4 Footprint

The RSI_VSMMU_GET_INFO command does not have any footprint.

DRAFT

B5.4 RSI types

This section defines types which are used in the RSI interface.

B5.4.1 RsiBoolean type

The RsiBoolean enumeration represents a boolean value.

The RsiBoolean enumeration is a [concrete type](#).

The width of the RsiBoolean enumeration is 1 bits.

The values of the RsiBoolean enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_FALSE	False
1	RSI_TRUE	True

B5.4.2 RsiCommandReturnCode type

The RsiCommandReturnCode enumeration represents a return code from an RSI command.

The RsiCommandReturnCode enumeration is a [concrete type](#).

The width of the RsiCommandReturnCode enumeration is 64 bits.

See also:

- [Chapter B1 Commands](#)

The values of the RsiCommandReturnCode enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_SUCCESS	Command completed successfully
1	RSI_ERROR_INPUT	The value of a command input value caused the command to fail
2	RSI_ERROR_STATE	The state of the current Realm or current REC does not match the state expected by the command
3	RSI_INCOMPLETE	The operation requested by the command is not complete
4	RSI_ERROR_UNKNOWN	The operation requested by the command failed for an unknown reason
5	RSI_ERROR_DEVICE	The state of a Realm device does not match the state expected by the command

Unused encodings for the RsiCommandReturnCode enumeration are reserved for use by future versions of this specification.

B5.4.3 RsiDeviceAttestationType type

The RsiDeviceAttestationType enumeration represents attestation type of a device.

The RsiDeviceAttestationType enumeration is a [concrete type](#).

The width of the RsiDeviceAttestationType enumeration is 64 bits.

See also:

- [A9.1.2 Device attestation types](#)

The values of the RsiDeviceAttestationType enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_INDEPENDENTLY_ATTESTED	Device is independently-attested.
1	RSI_PLATFORM_ATTESTED	Device is platform-attested.

Unused encodings for the RsiDeviceAttestationType enumeration are reserved for use by future versions of this specification.

The RsiDeviceAttestationType enumeration is used in the following types:

- [RsiDeviceInfo](#)

B5.4.4 RsiDeviceInfo type

The RsiDeviceInfo structure contains device configuration information.

The RsiDeviceInfo structure is a [concrete type](#).

The width of the RsiDeviceInfo structure is 512 (0x200) bytes.

See also:

- [A9.5 Realm management of an assigned device interface](#)
- [B5.3.16 RSI_RDEV_GET_INFO command](#)

The members of the RsiDeviceInfo structure are shown in the following table.

Name	Byte offset	Type	Description
attest_type	0x0	RsiDeviceAttestationType	Attestation type
cert_id	0x8	UInt64	Certificate identifier
hash_algo	0x10	RsiHashAlgorithm	Algorithm used to generate device digests
cert_digest	0x40	Bits512	Certificate digest
meas_digest	0x80	Bits512	Measurement block digest
report_digest	0xc0	Bits512	Interface report digest

Unused bits of the RsiDeviceInfo structure MBZ.

B5.4.5 RsiDeviceState type

The RsiDeviceState enumeration represents state of an assigned Realm device.

The RsiDeviceState enumeration is a [concrete type](#).

The width of the RsiDeviceState enumeration is 64 bits.

See also:

- [A9.5 Realm management of an assigned device interface](#)

The values of the RsiDeviceState enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_RDEV_UNLOCKED	Device interface is unlocked.
1	RSI_RDEV_UNLOCKED_BUSY	Device interface is unlocked and is handling an interruptible Realm device operation.
2	RSI_RDEV_LOCKED	Device interface is locked.
3	RSI_RDEV_LOCKED_BUSY	Device interface is locked and is handling an interruptible Realm device operation.
4	RSI_RDEV_STARTED	Device interface is started.
5	RSI_RDEV_STARTED_BUSY	Device interface is started and is handling an interruptible Realm device operation.
6	RSI_RDEV_STOPPING	Device interface is stopping.
7	RSI_RDEV_STOPPED	Device interface is stopped.
8	RSI_RDEV_ERROR	Device interface has reported a fatal error.

Unused encodings for the RsiDeviceState enumeration are reserved for use by future versions of this specification.

B5.4.6 RsiDevMemCoherent type

The RsiDevMemCoherent enumeration represents whether a device memory location is within the system coherent memory space.

The RsiDevMemCoherent enumeration is a [concrete type](#).

The width of the RsiDevMemCoherent enumeration is 1 bits.

The values of the RsiDevMemCoherent enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_DEV_MEM_NON_COHERENT	A device memory location is not within the system coherent memory space
1	RSI_DEV_MEM_COHERENT	A device memory location is within the system coherent memory space

The RsiDevMemCoherent enumeration is used in the following types:

- [RsiDevMemFlags](#)

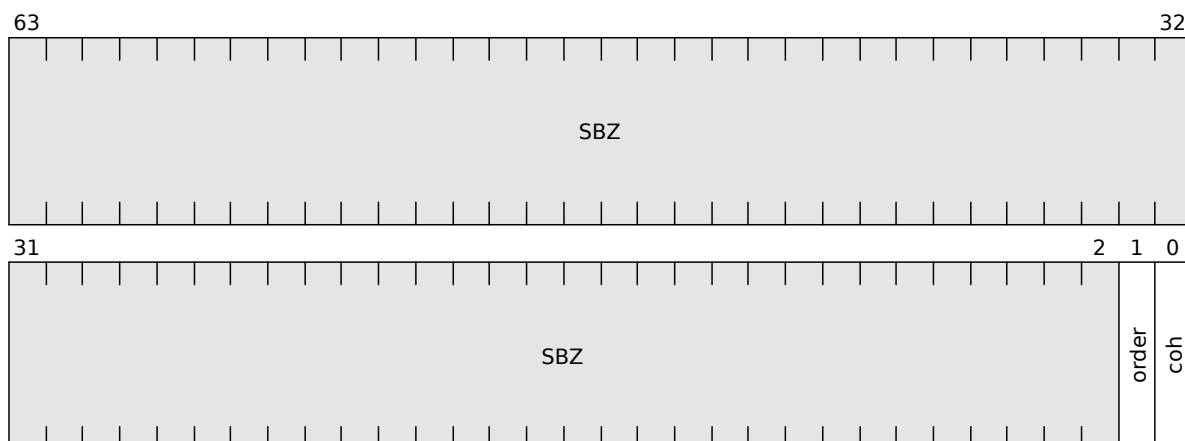
B5.4.7 RsiDevMemFlags type

The RsiDevMemFlags fieldset contains flags which describe properties of a device memory mapping.

The RsiDevMemFlags fieldset is a [concrete type](#).

The width of the RsiDevMemFlags fieldset is 64 bits.

The fields of the RsiDevMemFlags fieldset are shown in the following diagram.



The fields of the RsiDevMemFlags fieldset are shown in the following table.

Name	Bits	Description	Value
coh	0	Whether the output address of the device memory mapping is within the system coherent memory space.	RsiDevMemCoherent
order	1	Ordering properties of the device memory location.	RsiDevMemOrdering
	63:2	Reserved	SBZ

B5.4.8 RsiDevMemOrdering type

The RsiDevMemOrdering enumeration represents ordering properties of a device memory location.

The RsiDevMemOrdering enumeration is a [concrete type](#).

The width of the RsiDevMemOrdering enumeration is 1 bits.

The values of the RsiDevMemOrdering enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_DEV_MEM_NOT_LIMITED_ORDER	A device memory location is not within a Limited Order Region (LOR)
1	RSI_DEV_MEM_LIMITED_ORDER	A device memory location is within a Limited Order Region (LOR)

The RsiDevMemOrdering enumeration is used in the following types:

- [RsiDevMemFlags](#)

B5.4.9 RsiFeature type

The RsiFeature enumeration represents whether a feature is enabled.

The RsiFeature enumeration is a [concrete type](#).

The width of the RsiFeature enumeration is 1 bits.

The values of the RsiFeature enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_FEATURE_FALSE	Feature is not enabled.
1	RSI_FEATURE_TRUE	Feature is enabled.

The RsiFeature enumeration is used in the following types:

- [RsiFeatureRegister0](#)

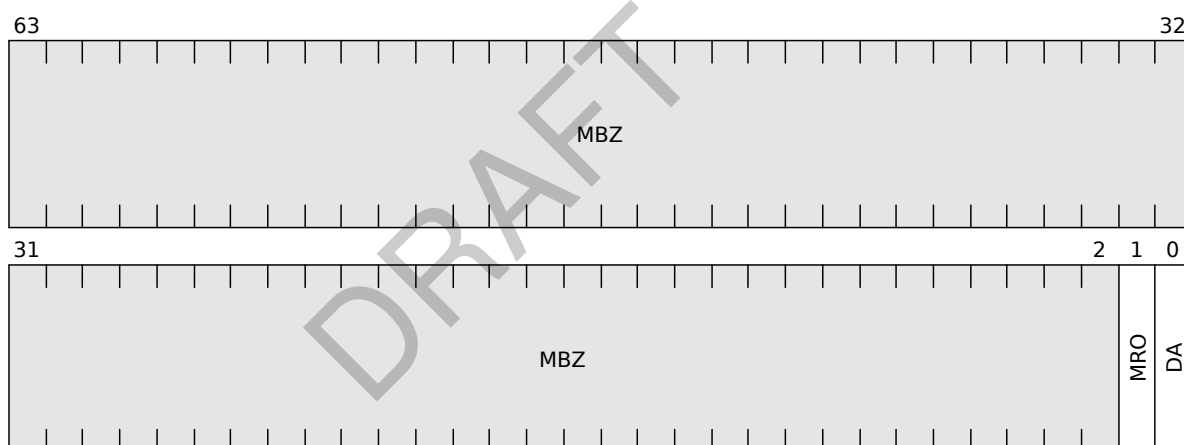
B5.4.10 RsiFeatureRegister0 type

The RsiFeatureRegister0 fieldset contains RSI feature register 0.

The RsiFeatureRegister0 fieldset is a [concrete type](#).

The width of the RsiFeatureRegister0 fieldset is 64 bits.

The fields of the RsiFeatureRegister0 fieldset are shown in the following diagram.



The fields of the RsiFeatureRegister0 fieldset are shown in the following table.

Name	Bits	Description	Value
DA	0	Whether Realm device assignment is supported	RsiFeature
MRO	1	Whether “mostly read-only” permissions are supported	RsiFeature
	63:2	Reserved	MBZ

B5.4.11 RsiGicOwner type

The RsiGicOwner enumeration represents which Plane is GIC owner.

The RsiGicOwner enumeration is a [concrete type](#).

The width of the RsiGicOwner enumeration is 1 bits.

The values of the RsiGicOwner enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_GIC_OWNER_0	Plane 0 is GIC owner.
1	RSI_GIC_OWNER_N	Plane N is GIC owner.

The RsiGicOwner enumeration is used in the following types:

- [RsiPlaneEnterFlags](#)

B5.4.12 RsiHashAlgorithm type

The RsiHashAlgorithm enumeration represents hash algorithm.

The RsiHashAlgorithm enumeration is a [concrete type](#).

The width of the RsiHashAlgorithm enumeration is 8 bits.

See also:

- [B5.3.25 RSI_REALM_CONFIG command](#)

The values of the RsiHashAlgorithm enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_HASH_SHA_256	SHA-256 (Secure Hash Standard (SHS) [20])
1	RSI_HASH_SHA_512	SHA-512 (Secure Hash Standard (SHS) [20])

Unused encodings for the RsiHashAlgorithm enumeration are reserved for use by future versions of this specification.

The RsiHashAlgorithm enumeration is used in the following types:

- [RsiDeviceInfo](#)
- [RsiRealmConfig](#)

B5.4.13 RsiHostCall type

The RsiHostCall structure contains data structure used to pass Host call arguments and return values.

The RsiHostCall structure is a [concrete type](#).

The width of the RsiHostCall structure is 256 (0x100) bytes.

See also:

- [A4.5 Host call](#)
- [B5.3.4 RSI_HOST_CALL command](#)

The members of the RsiHostCall structure are shown in the following table.

Name	Byte offset	Type	Description
imm	0x0	UInt16	Immediate value
gprs[31]	0x8	Bits64	Registers

Unused bits of the RsiHostCall structure SBZ.

B5.4.14 RsiInterfaceVersion type

The RsiInterfaceVersion fieldset contains an RSI interface version.

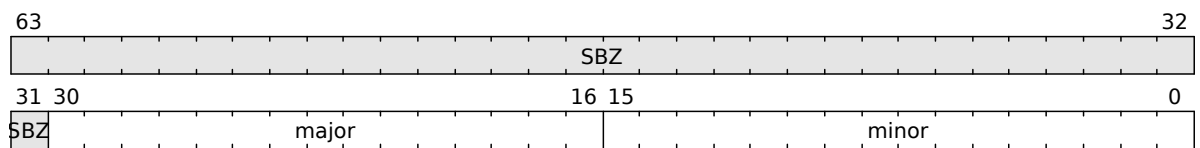
The RsiInterfaceVersion fieldset is a [concrete type](#).

The width of the RsiInterfaceVersion fieldset is 64 bits.

See also:

- [B5.1 RSI version](#)
- [B5.3.26 RSI_VERSION command](#)

The fields of the RsiInterfaceVersion fieldset are shown in the following diagram.



The fields of the RsiInterfaceVersion fieldset are shown in the following table.

Name	Bits	Description	Value
minor	15:0	Interface minor version number (the value y in interface version $x.y$)	UInt16
major	30:16	Interface major version number (the value x in interface version $x.y$)	UInt15
	63:31	Reserved	SBZ

B5.4.15 RsiPlaneEnter type

The RsiPlaneEnter structure contains data passed from P0 to the RMM on Plane entry.

The RsiPlaneEnter structure is a [concrete type](#).

The width of the RsiPlaneEnter structure is 2048 (0×800) bytes.

The members of the RsiPlaneEnter structure are shown in the following table.

Name	Byte offset	Type	Description
flags	0×0	RsiPlaneEnterFlags	Flags
pc	0×8	Bits64	Program counter
gprs[31]	0×100	Bits64	Registers
gicv3_hcr	0×200	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0×208	Bits64	GICv3 List Register values

Unused bits of the RsiPlaneEnter structure SBZ.

The RsiPlaneEnter structure is used in the following types:

- [RsiPlaneRun](#)

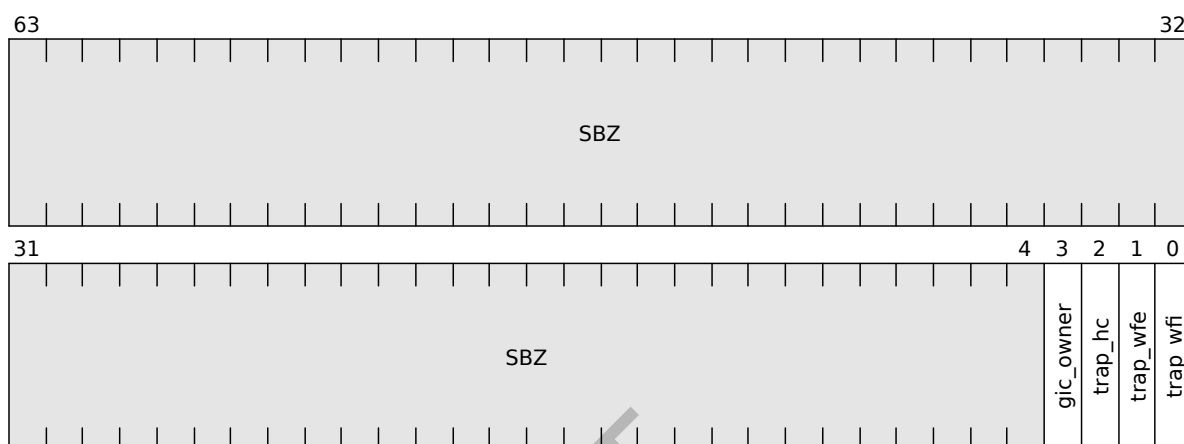
B5.4.16 RsiPlaneEnterFlags type

The RsiPlaneEnterFlags fieldset contains flags provided by P0 during Plane entry.

The RsiPlaneEnterFlags fieldset is a [concrete type](#).

The width of the RsiPlaneEnterFlags fieldset is 64 bits.

The fields of the RsiPlaneEnterFlags fieldset are shown in the following diagram.



The fields of the RsiPlaneEnterFlags fieldset are shown in the following table.

Name	Bits	Description	Value
trap_wfi	0	Whether to trap WFI execution by the Plane.	RsiTrap
trap_wfe	1	Whether to trap WFE execution by the Plane.	RsiTrap
trap_hc	2	Whether to trap RSI_HOST_CALL execution by the Plane. RSI_TRAP: execution of RSI_HOST_CALL causes Plane exit RSI_NO_TRAP: execution of RSI_HOST_CALL causes REC exit to Host	RsiTrap
gic_owner	3	Whether to transfer GIC ownership to the target Plane.	RsiGicOwner
	63:4	Reserved	SBZ

The RsiPlaneEnterFlags fieldset is used in the following types:

- [RsiPlaneEnter](#)

B5.4.17 RsiPlaneExit type

The RsiPlaneExit structure contains data passed from the RMM to P0 on Plane exit.

The RsiPlaneExit structure is a [concrete type](#).

The width of the RsiPlaneExit structure is 2048 (0x800) bytes.

The members of the RsiPlaneExit structure are shown in the following table.

Name	Byte offset	Type	Description
reason	0x0	RsiPlaneExitReason	Exit reason
elr_el2	0x100	Bits64	Exception Link Register
esr_el2	0x108	Bits64	Exception Syndrome Register
far_el2	0x110	Bits64	Fault Address Register
hpfar_el2	0x118	Bits64	Hypervisor IPA Fault Address register
gprs[31]	0x200	Bits64	Registers
gicv3_hcr	0x300	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0x308	Bits64	GICv3 List Register values
gicv3_misr	0x388	Bits64	GICv3 Maintenance Interrupt State Register value
gicv3_vmcr	0x390	Bits64	GICv3 Virtual Machine Control Register value
cntp_ctl	0x400	Bits64	Counter-timer Physical Timer Control Register value
cntp_cval	0x408	Bits64	Counter-timer Physical Timer CompareValue Register value
cntv_ctl	0x410	Bits64	Counter-timer Virtual Timer Control Register value
cntv_cval	0x418	Bits64	Counter-timer Virtual Timer CompareValue Register value

Unused bits of the RsiPlaneExit structure SBZ.

The RsiPlaneExit structure is used in the following types:

- [RsiPlaneRun](#)

B5.4.18 RsiPlaneExitReason type

The RsiPlaneExitReason enumeration represents the reason for a Plane exit.

The RsiPlaneExitReason enumeration is a [concrete type](#).

The width of the RsiPlaneExitReason enumeration is 8 bits.

The values of the RsiPlaneExitReason enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_EXIT_SYNC	Plane exit due to synchronous exception

Unused encodings for the RsiPlaneExitReason enumeration are reserved for use by future versions of this specification.

The RsiPlaneExitReason enumeration is used in the following types:

- [RsiPlaneExit](#)

B5.4.19 RsiPlaneRun type

The RsiPlaneRun structure contains fields used to share information between RMM and P0 during Plane entry and Plane exit.

The RsiPlaneRun structure is a [concrete type](#).

The width of the RsiPlaneRun structure is 4096 (0x1000) bytes.

The members of the RsiPlaneRun structure are shown in the following table.

Name	Byte offset	Type	Description
enter	0x0	RsiPlaneEnter	Entry information
exit	0x800	RsiPlaneExit	Exit information

B5.4.20 RsiRealmConfig type

The RsiRealmConfig structure contains realm configuration.

The RsiRealmConfig structure is a [concrete type](#).

The width of the RsiRealmConfig structure is 4096 (0x1000) bytes.

See also:

- [B5.3.25 RSI_REALM_CONFIG command](#)

The members of the RsiRealmConfig structure are shown in the following table.

Name	Byte offset	Type	Description
ipa_width	0x0	UInt64	IPA width in bits
hash_algo	0x8	RsiHashAlgorithm	Hash algorithm
num_aux_planes	0x10	UInt64	Number of auxiliary Planes
gicv3_vtr	0x18	Bits64	GICv3 VGIC Type Register value
rpv	0x200	Bits512	Realm Personalization Value

Unused bits of the RsiRealmConfig structure MBZ.

B5.4.21 RsiResponse type

The RsiResponse enumeration represents whether the Host accepted or rejected a Realm request.

The RsiResponse enumeration is a [concrete type](#).

The width of the RsiResponse enumeration is 1 bits.

The values of the RsiResponse enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_ACCEPT	Host accepted the Realm request.
1	RSI_REJECT	Host rejected the Realm request.

B5.4.22 RsiRipas type

The RsiRipas enumeration represents realm IPA state.

The RsiRipas enumeration is a [concrete type](#).

The width of the RsiRipas enumeration is 8 bits.

See also:

- [A5.4 RIPAS change](#)
- [B5.3.5 RSI_IPA_STATE_GET command](#)
- [B5.3.6 RSI_IPA_STATE_SET command](#)

The values of the RsiRipas enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_EMPTY	Address where no Realm resources are mapped.
1	RSI_RAM	Address where private code or data owned by the Realm is mapped.
2	RSI_DESTROYED	Address which is inaccessible to the Realm due to an action taken by the Host.
3	RSI_DEV	Address where memory of an assigned Realm device is mapped.

Unused encodings for the RsiRipas enumeration are reserved for use by future versions of this specification.

B5.4.23 RsiRipasChangeDestroyed type

The RsiRipasChangeDestroyed enumeration represents whether a RIPAS change from DESTROYED should be permitted.

The RsiRipasChangeDestroyed enumeration is a [concrete type](#).

The width of the RsiRipasChangeDestroyed enumeration is 1 bits.

The values of the RsiRipasChangeDestroyed enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_NO_CHANGE_DESTROYED	A RIPAS change from DESTROYED should not be permitted.
1	RSI_CHANGE_DESTROYED	A RIPAS change from DESTROYED should be permitted.

The RsiRipasChangeDestroyed enumeration is used in the following types:

- [RsiRipasChangeFlags](#)

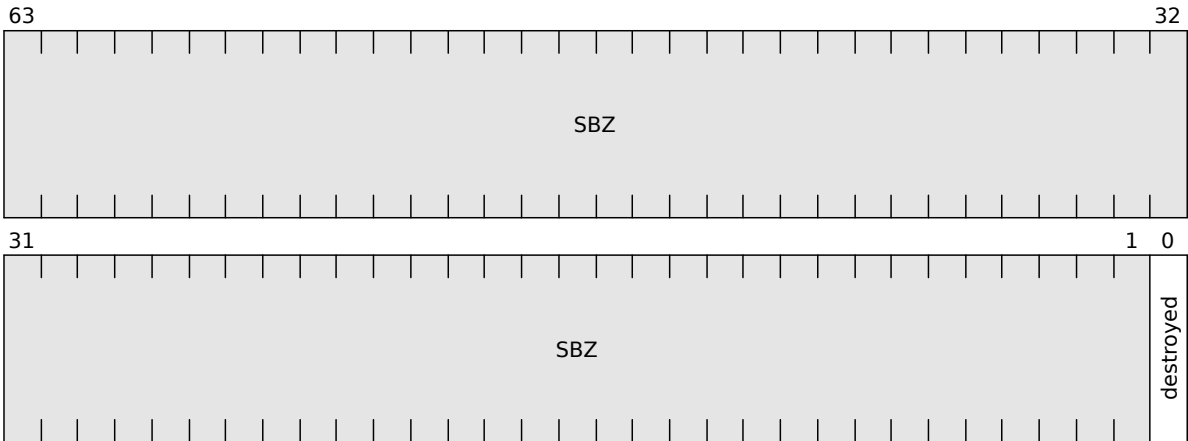
B5.4.24 RsiRipasChangeFlags type

The RsiRipasChangeFlags fieldset contains flags provided by the Realm when requesting a RIPAS change.

The RsiRipasChangeFlags fieldset is a [concrete type](#).

The width of the RsiRipasChangeFlags fieldset is 64 bits.

The fields of the RsiRipasChangeFlags fieldset are shown in the following diagram.



The fields of the RsiRipasChangeFlags fieldset are shown in the following table.

Name	Bits	Description	Value
destroyed	0	Whether a RIPAS change from DESTROYED should be permitted	RsiRipasChangeDestroyed
	63:1	Reserved	SBZ

B5.4.25 RsiTrap type

The RsiTrap enumeration represents whether a trap is enabled.

The RsiTrap enumeration is a [concrete type](#).

The width of the RsiTrap enumeration is 1 bits.

The values of the RsiTrap enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_NO_TRAP	Trap is disabled.
1	RSI_TRAP	Trap is enabled.

The RsiTrap enumeration is used in the following types:

- [RsiPlaneEnterFlags](#)

Chapter B6

Power State Control Interface

This section describes how Power State Control Interface (PSCI) function execution by a Realm execution of SMC instructions is handled.

B6.1 PSCI overview

I_{GBVWX}

In this section,

- `rec` refers to the currently executing REC
- `exit` refer to the `RmiRecExit` object which was provided to the `RMI_REC_ENTER` command
- `target_rec` refers to the REC object identified by an MPIDR value passed to a PSCI function.

I_{GHKCJ}

The RMM provides a trusted implementation of parts of the PSCI ABI. This section describes the checks performed by the RMM when a Realm executes a PSCI command, and the internal RMM state changes which result from a successful PSCI command execution. Successful execution by the RMM of some PSCI commands results in a *REC exit due to PSCI*, which allows the Host to perform further processing of the command.

I_{XHDQF}

The HVC conduit for PSCI is not supported for Realms.

See also:

- [Arm Power State Coordination Interface \(PSCI\) \[23\]](#)
- [A2.3.2 REC attributes](#)
- [A4.3.7 REC exit due to PSCI](#)
- [A4.5 Host call](#)
- [D1.4 PSCI flows](#)

B6.2 PSCI version

R_{TFQVF}

The RMM must support version ≥ 1.1 of the Power State Control Interface.

See also:

- [B6.3.8 PSCI_VERSION command](#)

B6.3 PSCI commands

The following table summarizes the FIDs of commands in the PSCI interface.

FID	Command
0x84000000	PSCI_VERSION
...	
0x84000002	PSCI_CPU_OFF
...	
0x84000008	PSCI_SYSTEM_OFF
0x84000009	PSCI_SYSTEM_RESET
0x8400000A	PSCI_FEATURES
...	
0xC4000001	PSCI_CPU_SUSPEND
...	
0xC4000003	PSCI_CPU_ON
0xC4000004	PSCI_AFFINITY_INFO

B6.3.1 PSCI_AFFINITY_INFO command

Query status of a VPE.

This command causes a REC exit due to PSCI. In response, the Host should provide the target REC (identified by `target_affinity`) by calling `RMI_PSCI_COMPLETE`.

See also:

- [A2.3.2 REC attributes](#)
- [A4.3.7 REC exit due to PSCI](#)
- [B4.3.21 RMI_PSCI_COMPLETE command](#)
- [B6.3.2 PSCI_CPU_OFF command](#)
- [B6.3.3 PSCI_CPU_ON command](#)

B6.3.1.1 Interface

B6.3.1.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000004
target_affinity	X1	63:0	Bits64	This parameter contains a copy of the affinity fields of the MPIDR register
lowest_affinity_level	X2	31:0	UInt32	Denotes the lowest affinity level field that is valid in the target_affinity parameter

The following unused bits of PSCI_AFFINITY_INFO input values SBZ: X2[63:32].

B6.3.1.1.2 Context

The PSCI_AFFINITY_INFO command operates on the following context.

Name	Type	Value	Before	Description
target_rec	RmmRec	RecFromMpidr(target_affinity)	false	Target REC

B6.3.1.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	PsciReturnCode	Command return code

B6.3.1.2 Failure conditions

ID	Condition
target_bound	pre: lowest_affinity_level != 0 post: result == PSCI_INVALID_PARAMETERS

ID	Condition
target_match	pre: !MpidrIsUsed(target_affinity) post: result == PSCI_INVALID_PARAMETERS

B6.3.1.2.1 Failure condition ordering

The PSCI_AFFINITY_INFO command does not have any failure condition orderings.

B6.3.1.3 Success conditions

ID	Condition
runnable	pre: target_rec.flags.runnable == RUNNABLE post: result == PSCI_SUCCESS
not_runnable	pre: target_rec.flags.runnable == NOT_RUNNABLE post: result == PSCI_OFF

B6.3.1.4 Footprint

The PSCI_AFFINITY_INFO command does not have any footprint.

B6.3.2 PSCI_CPU_OFF command

Power down the calling core.

This command causes a REC exit due to PSCI.

See also:

- [A2.3.2 REC attributes](#)
- [A4.3.7 REC exit due to PSCI](#)
- [B6.3.3 PSCI_CPU_ON command](#)
- [B6.3.4 PSCI_CPU_SUSPEND command](#)

B6.3.2.1 Interface

B6.3.2.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0x84000002

B6.3.2.1.2 Context

The PSCI_CPU_OFF command operates on the following context.

Name	Type	Value	Before	Description
rec	RmmRec	CurrentRec()	false	Current REC

B6.3.2.1.3 Output values

The PSCI_CPU_OFF command does not have any output values.

Following execution of PSCI_CPU_OFF, control does not return to the caller.

B6.3.2.2 Failure conditions

The PSCI_CPU_OFF command does not have any failure conditions.

B6.3.2.3 Success conditions

The PSCI_CPU_OFF command does not have any success conditions.

Following execution of PSCI_CPU_OFF, control does not return to the caller.

B6.3.2.4 Footprint

The PSCI_CPU_OFF command does not have any footprint.

B6.3.3 PSCI_CPU_ON command

Power up a core.

This command causes a REC exit due to PSCI. In response, the Host should provide the target REC (identified by `target_cpu`) by calling `RMI_PSCI_COMPLETE`.

See also:

- [A2.3.2 REC attributes](#)
- [A4.3.7 REC exit due to PSCI](#)
- [B4.3.21 RMI_PSCI_COMPLETE command](#)
- [B6.3.2 PSCI_CPU_OFF command](#)
- [B6.3.4 PSCI_CPU_SUSPEND command](#)
- [D1.4.1 PSCI_CPU_ON flow](#)

B6.3.3.1 Interface

B6.3.3.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000003
target_cpu	X1	63:0	Bits64	This parameter contains a copy of the affinity fields of the MPIDR register
entry_point_address	X2	63:0	Address	Address at which the core must resume execution
context_id	X3	31:0	UInt32	This parameter is only meaningful to the caller (must be present in X0 of the target PE upon first entry to Non-Secure exception level)

The following unused bits of PSCI_CPU_ON input values SBZ: X3[63:32].

B6.3.3.1.2 Context

The PSCI_CPU_ON command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm
target_rec	RmmRec	RecFromMpidr(target_cpu)	false	Target REC

B6.3.3.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	PsciReturnCode	Command return code

B6.3.3.2 Failure conditions

ID	Condition
entry	pre: !AddrIsProtected(entry_point_address, realm) post: result == PSCI_INVALID_ADDRESS
mpidr	pre: !MpidrIsUsed(target_cpu) post: result == PSCI_INVALID_PARAMETERS
runnable	pre: target_rec.flags.runnable == RUNNABLE post: result == PSCI_ALREADY_ON

B6.3.3.2.1 Failure condition ordering

The PSCI_CPU_ON command does not have any failure condition orderings.

B6.3.3.3 Success conditions

ID	Condition
entry	target_rec.pc == ToBits64(UInt(entry_point_address))
runnable	target_rec.flags.runnable == RUNNABLE

B6.3.3.4 Footprint

ID	Value
runnable	target_rec.flags.runnable

B6.3.4 PSCI_CPU_SUSPEND command

Suspend execution on the calling VPE.

This command causes a REC exit due to PSCI.

See also:

- [A4.3.7 REC exit due to PSCI](#)
- [B6.3.2 PSCI_CPU_OFF command](#)
- [B6.3.3 PSCI_CPU_ON command](#)

B6.3.4.1 Interface

B6.3.4.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0xC4000001
power_state	X1	31:0	UInt32	Identifier for a specific local state
entry_point_address	X2	63:0	Address	Address at which the core must resume execution
context_id	X3	63:0	UInt64	This parameter is only meaningful to the caller (must be present in X0 upon first entry to Non-Secure exception level)

The following unused bits of PSCI_CPU_SUSPEND input values SBZ: X1[63:32].

The RMM treats all target power states as suspend requests, and therefore the `entry_point_address` and `context_id` arguments are ignored.

B6.3.4.1.2 Output values

The PSCI_CPU_SUSPEND command does not have any output values.

Following execution of PSCI_CPU_SUSPEND, control does not return to the caller.

B6.3.4.2 Failure conditions

The PSCI_CPU_SUSPEND command does not have any failure conditions.

B6.3.4.3 Success conditions

The PSCI_CPU_SUSPEND command does not have any success conditions.

Following execution of PSCI_CPU_SUSPEND, control does not return to the caller.

B6.3.4.4 Footprint

The PSCI_CPU_SUSPEND command does not have any footprint.

B6.3.5 PSCI_FEATURES command

Query whether a specific PSCI feature is implemented.

See also:

- [B6.3.1 PSCI_AFFINITY_INFO command](#)
- [B6.3.2 PSCI_CPU_OFF command](#)
- [B6.3.3 PSCI_CPU_ON command](#)
- [B6.3.4 PSCI_CPU_SUSPEND command](#)
- [B6.3.6 PSCI_SYSTEM_OFF command](#)
- [B6.3.7 PSCI_SYSTEM_RESET command](#)

B6.3.5.1 Interface

B6.3.5.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0x8400000A
psci_func_id	X1	31:0	UInt32	Function ID for a PSCI Function

The following unused bits of PSCI_FEATURES input values SBZ: X1[63:32].

B6.3.5.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	PsciReturnCode	Command return code

B6.3.5.2 Failure conditions

The PSCI_FEATURES command does not have any failure conditions.

B6.3.5.3 Success conditions

ID	Condition
func_ok	pre: psci_func_id is a supported PSCI function. post: result == PSCI_SUCCESS
func_not_ok	pre: psci_func_id is not a supported PSCI function. post: result == PSCI_NOT_SUPPORTED

B6.3.5.4 Footprint

The PSCI_FEATURES command does not have any footprint.

B6.3.6 PSCI_SYSTEM_OFF command

Shut down the system.

This command causes a REC exit due to PSCI.

See also:

- [A2.3.2 REC attributes](#)
- [A4.3.7 REC exit due to PSCI](#)
- [B6.3.7 PSCI_SYSTEM_RESET command](#)

B6.3.6.1 Interface

B6.3.6.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0x84000008

B6.3.6.1.2 Context

The PSCI_SYSTEM_OFF command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B6.3.6.1.3 Output values

The PSCI_SYSTEM_OFF command does not have any output values.

Following execution of PSCI_SYSTEM_OFF, control does not return to the caller.

B6.3.6.2 Failure conditions

The PSCI_SYSTEM_OFF command does not have any failure conditions.

B6.3.6.3 Success conditions

ID	Condition
state	realm.state == REALM_SYSTEM_OFF

Following execution of PSCI_SYSTEM_OFF, control does not return to the caller.

B6.3.6.4 Footprint

The PSCI_SYSTEM_OFF command does not have any footprint.

B6.3.7 PSCI_SYSTEM_RESET command

Shut down the system.

This command causes a REC exit due to PSCI.

See also:

- [A2.3.2 REC attributes](#)
- [A4.3.7 REC exit due to PSCI](#)
- [B6.3.6 PSCI_SYSTEM_OFF command](#)

B6.3.7.1 Interface

B6.3.7.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0x84000009

B6.3.7.1.2 Context

The PSCI_SYSTEM_RESET command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	CurrentRealm()	false	Current Realm

B6.3.7.1.3 Output values

The PSCI_SYSTEM_RESET command does not have any output values.

Following execution of PSCI_SYSTEM_RESET, control does not return to the caller.

B6.3.7.2 Failure conditions

The PSCI_SYSTEM_RESET command does not have any failure conditions.

B6.3.7.3 Success conditions

ID	Condition
state	realm.state == REALM_SYSTEM_OFF

Following execution of PSCI_SYSTEM_RESET, control does not return to the caller.

B6.3.7.4 Footprint

The PSCI_SYSTEM_RESET command does not have any footprint.

B6.3.8 PSCI_VERSION command

Query the version of PSCI implemented.

B6.3.8.1 Interface

B6.3.8.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value 0x84000000

B6.3.8.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	PsciInterfaceVersion	Interface version

See also:

- [B6.2 PSCI version](#)

B6.3.8.2 Failure conditions

The PSCI_VERSION command does not have any failure conditions.

B6.3.8.3 Success conditions

The PSCI_VERSION command does not have any success conditions.

B6.3.8.4 Footprint

The PSCI_VERSION command does not have any footprint.

B6.4 PSCI types

This section defines types which are used in the PSCI interface.

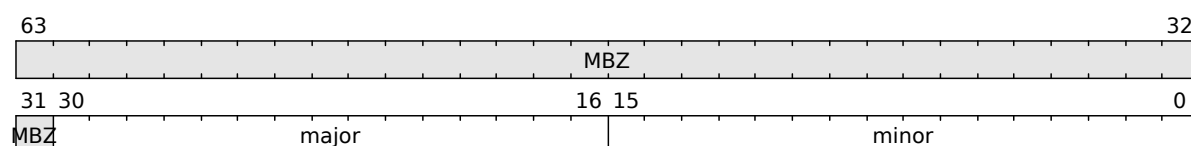
B6.4.1 PsciInterfaceVersion type

The PsciInterfaceVersion fieldset contains an PSCI interface version.

The PsciInterfaceVersion fieldset is a [concrete type](#).

The width of the PsciInterfaceVersion fieldset is 64 bits.

The fields of the PsciInterfaceVersion fieldset are shown in the following diagram.



The fields of the PsciInterfaceVersion fieldset are shown in the following table.

Name	Bits	Description	Value
minor	15:0	Interface minor version number (the value y in interface version $x.y$)	UInt16
major	30:16	Interface major version number (the value x in interface version $x.y$)	UInt15
	63:31	Reserved	MBZ

B6.4.2 PsciReturnCode type

The PsciReturnCode enumeration represents the return code of a PSCI command.

The PsciReturnCode enumeration is a [concrete type](#).

The width of the PsciReturnCode enumeration is 64 bits.

The values of the PsciReturnCode enumeration are shown in the following table.

Encoding	Name	Description
-9	PSCI_INVALID_ADDRESS	Refer to PSCI specification
-8	PSCI_DISABLED	Refer to PSCI specification
-7	PSCI_NOT_PRESENT	Refer to PSCI specification
-6	PSCI_INTERNAL_FAILURE	Refer to PSCI specification
-5	PSCI_ON_PENDING	Refer to PSCI specification
-4	PSCI_ALREADY_ON	Refer to PSCI specification
-3	PSCI_DENIED	Refer to PSCI specification
-2	PSCI_INVALID_PARAMETERS	Refer to PSCI specification
-1	PSCI_NOT_SUPPORTED	Refer to PSCI specification

Encoding	Name	Description
0	PSCI_SUCCESS	Refer to PSCI specification
1	PSCI_OFF	Refer to PSCI specification

Unused encodings for the PsciReturnCode enumeration are reserved for use by future versions of this specification.

DRAFT

DRAFT

Part C

Constants and types

Chapter C1

RMM constants

This section describes constants which are used in the definition of RMM commands or RMM abstract state.

C1.1 RMM_GRANULE_SIZE

Size of a Granule in bytes.

The value of RMM_GRANULE_SIZE is 0x1000.

C1.2 RMM_NUM_PERM_OVERLAY_INDICES

Number of permission overlay indices.

The value of RMM_NUM_PERM_OVERLAY_INDICES is 15.

C1.3 RMM_RTT_BLOCK_LEVEL

RTT level of a block entry.

The value of RMM_RTT_BLOCK_LEVEL is 2.

C1.4 RMM_RTT_PAGE_LEVEL

RTT level of a page entry.

The value of RMM_RTT_PAGE_LEVEL is 3.

C1.5 RMM_RTT_TREE_PRIMARY

Index of primary RTT tree.

The value of RMM_RTT_TREE_PRIMARY is 0.

DRAFT

Chapter C2

RMM types

This section describes types which are used to model the abstract state of the RMM.

C2.1 RmmAddressRange type

The RmmAddressRange structure contains address range.

The RmmAddressRange structure is an [abstract type](#).

The members of the RmmAddressRange structure are shown in the following table.

Name	Type	Description
base	Address	Base of address range (inclusive)
top	Address	Top of address range (exclusive)

The RmmAddressRange structure is used in the following types:

- [RmmPdev](#)

C2.2 RmmBoolean type

The RmmBoolean enumeration represents whether a feature is enabled.

The RmmBoolean enumeration is an [abstract type](#).

The values of the RmmBoolean enumeration are shown in the following table.

Name	Description
RMM_FALSE	False
RMM_TRUE	True

The RmmBoolean enumeration is used in the following types:

- [RmmRec](#)
- [RmmRttS2APDirect](#)
- [RmmRttWalkNotAligned](#)

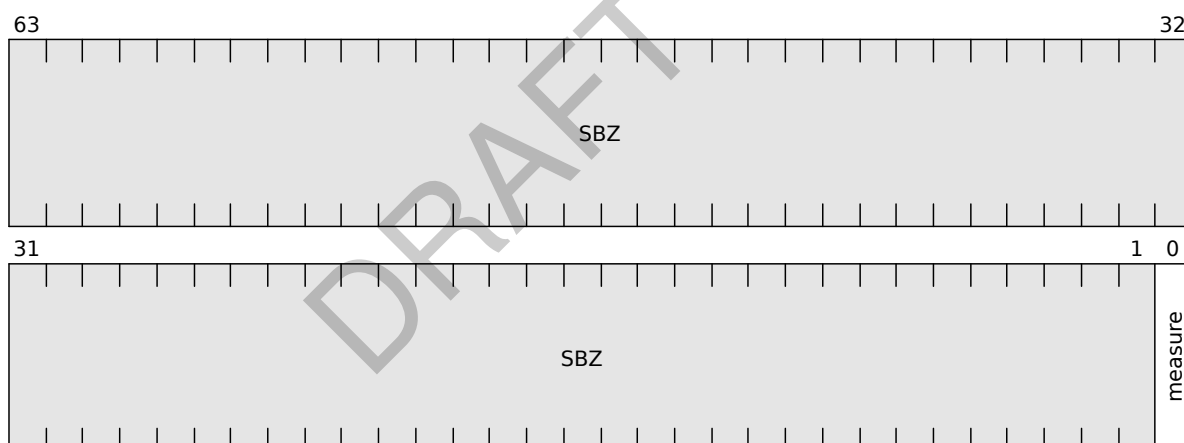
C2.3 RmmDataFlags type

The RmmDataFlags fieldset contains flags provided by the Host during DATA Granule creation.

The RmmDataFlags fieldset is a [concrete type](#).

The width of the RmmDataFlags fieldset is 64 bits.

The fields of the RmmDataFlags fieldset are shown in the following diagram.



The fields of the RmmDataFlags fieldset are shown in the following table.

Name	Bits	Description	Value
measure	0	Whether to measure DATA Granule contents	RmmDataMeasureContent
	63:1	Reserved	SBZ

The RmmDataFlags fieldset is used in the following types:

- [RmmMeasurementDescriptorData](#)

C2.4 RmmDataMeasureContent type

The RmmDataMeasureContent enumeration represents whether to measure DATA Granule contents.

The RmmDataMeasureContent enumeration is a [concrete type](#).

The width of the RmmDataMeasureContent enumeration is 1 bits.

The values of the RmmDataMeasureContent enumeration are shown in the following table.

Encoding	Name	Description
0	NO_MEASURE_CONTENT	Do not measure DATA Granule contents.
1	MEASURE_CONTENT	Measure DATA Granule contents.

The RmmDataMeasureContent enumeration is used in the following types:

- [RmmDataFlags](#)

C2.5 RmmDevCommState type

The RmmDevCommState enumeration represents the state of communication between an RMM device object and a device.

The RmmDevCommState enumeration is an [abstract type](#).

The values of the RmmDevCommState enumeration are shown in the following table.

Name	Description
DEV_COMM_ACTIVE	The RMM has initiated a device transaction. One or more device requests associated with this device transaction have been sent from the RMM to the device. The RMM has not received all the expected device responses associated with this device transaction.
DEV_COMM_ERROR	The RMM encountered an error during communication with the device.
DEV_COMM_IDLE	The RMM is not communicating with the device.
DEV_COMM_PENDING	The RMM has a device request which is ready to be sent to the device.

The RmmDevCommState enumeration is used in the following types:

- [RmmPdev](#)
- [RmmVdev](#)

C2.6 RmmDevMemCoherent type

The RmmDevMemCoherent enumeration represents whether a device memory location is within the system coherent memory space.

The RmmDevMemCoherent enumeration is an [abstract type](#).

The values of the RmmDevMemCoherent enumeration are shown in the following table.

Name	Description
DEV_MEM_COHERENT	A device memory location is within the system coherent memory space

Name	Description
DEV_MEM_NON_COHERENT	A device memory location is not within the system coherent memory space

The RmmDevMemCoherent enumeration is used in the following types:

- [RmmDevMemFlags](#)

C2.7 RmmDevMemFlags type

The RmmDevMemFlags structure contains flags which describe properties of a device memory mapping.

The RmmDevMemFlags structure is an [abstract type](#).

The members of the RmmDevMemFlags structure are shown in the following table.

Name	Type	Description
coh	RmmDevMemCoherent	Whether the output address of the device memory mapping is within the system coherent memory space.
order	RmmDevMemOrdering	Ordering properties of the device memory location.

The RmmDevMemFlags structure is used in the following types:

- [RmmRec](#)

C2.8 RmmDevMemOrdering type

The RmmDevMemOrdering enumeration represents ordering properties of a device memory location.

The RmmDevMemOrdering enumeration is an [abstract type](#).

The values of the RmmDevMemOrdering enumeration are shown in the following table.

Name	Description
DEV_MEM_LIMITED_ORDER	A device memory location is within a Limited Order Region (LOR)
DEV_MEM_NOT_LIMITED_ORDER	A device memory location is not within a Limited Order Region (LOR)

The RmmDevMemOrdering enumeration is used in the following types:

- [RmmDevMemFlags](#)

C2.9 RmmFeature type

The RmmFeature enumeration represents whether a feature is enabled.

The RmmFeature enumeration is an [abstract type](#).

See also:

- [Chapter A3 Feature discovery and configuration](#)

The values of the RmmFeature enumeration are shown in the following table.

Name	Description
FEATURE_FALSE	<ul style="list-style-type: none">• During discovery: Feature is not supported.• During selection: Feature is not enabled.
FEATURE_TRUE	<ul style="list-style-type: none">• During discovery: Feature is supported.• During selection: Feature is enabled.

The RmmFeature enumeration is used in the following types:

- [RmmFeatures](#)
- [RmmRealm](#)
- [RmmVdev](#)

C2.10 RmmFeatures type

The RmmFeatures structure contains features supported by RMM implementation.

The RmmFeatures structure is an [abstract type](#).

See also:

- [Chapter A3 Feature discovery and configuration](#)

The members of the RmmFeatures structure are shown in the following table.

Name	Type	Description
max_ipa_width	UInt64	Maximum IPA width
feat_lpa2	RmmFeature	Whether LPA2 is supported
feat_sve	RmmFeature	Whether SVE is supported
max_sve_vl	UInt64	Maximum SVE vector length
num_bps	UInt64	Number of breakpoints available
num_wps	UInt64	Number of watchpoints available
feat_pmu	RmmFeature	Number of watchpoints available
pmu_num_ctrs	UInt64	Number of PMU counters available
feat_sha_256	RmmFeature	Whether SHA-256 is supported
feat_sha_512	RmmFeature	Whether SHA-512 is supported
feat_da	RmmFeature	Whether Realm device assignment is supported
max_num_aux_planes	UInt64	Maximum number of auxiliary Planes
rtt_plane	RmmRttPlaneFeature	RTT usage models supported for multi-Plane Realms
rtt_s2ap_indirect	RmmFeature	Whether S2AP indirect encoding is supported for multi-Plane Realms

Name	Type	Description
max_mecid	Bits64	Maximum supported MECID
max_recs_order	UInt64	Order of the maximum number of RECs which can be created per Realm
gicv3_num_lrs	UInt64	Number of GICv3 List Registers which are available.

C2.11 RmmGptEntry type

The RmmGptEntry enumeration represents granule Protection Table entry.

The RmmGptEntry enumeration is an [abstract type](#).

See also:

- [B3.30 GranuleAccessPermitted function](#)

The values of the RmmGptEntry enumeration are shown in the following table.

Name	Description
GPT_AAP	Access permitted via any PAS.
GPT_NS	Access permitted via Non-secure PAS only.
GPT_REALM	Access permitted via Realm PAS only.
GPT_ROOT	Access permitted via Root PAS only.
GPT_SECURE	Access permitted via Secure PAS only.

The RmmGptEntry enumeration is used in the following types:

- [RmmGranule](#)

C2.12 RmmGranule type

The RmmGranule structure contains attributes of a Granule.

The RmmGranule structure is an [abstract type](#).

The members of the RmmGranule structure are shown in the following table.

Name	Type	Description
gpt	RmmGptEntry	GPT entry
state	RmmGranuleState	Lifecycle state

C2.13 RmmGranuleState type

The RmmGranuleState enumeration represents the state of a granule.

The RmmGranuleState enumeration is an [abstract type](#).

The values of the RmmGranuleState enumeration are shown in the following table.

Name	Description
DATA	Realm code or data.
DELEGATED	Delegated for use by the RMM.
DEV_MAPPED	Device memory, mapped into a Realm.
PDEV	Physical device.
PDEV_AUX	Physical device auxiliary Granule.
RD	Realm Descriptor.
REC	Realm Execution Context.
REC_AUX	Realm Execution Context auxiliary Granule.
RTT	Realm Translation Table.
UNDELEGATED	Not delegated for use by the RMM.
VDEV	Virtual device.
VDEV_AUX	Virtual device auxiliary Granule.
VSMMU	Virtual SMMU.

The RmmGranuleState enumeration is used in the following types:

- [RmmGranule](#)

C2.14 RmmHashAlgorithm type

The RmmHashAlgorithm enumeration represents hash algorithm.

The RmmHashAlgorithm enumeration is an [abstract type](#).

The values of the RmmHashAlgorithm enumeration are shown in the following table.

Name	Description
HASH_SHA_256	SHA-256 (Secure Hash Standard (SHS) [20])
HASH_SHA_512	SHA-512 (Secure Hash Standard (SHS) [20])

The RmmHashAlgorithm enumeration is used in the following types:

- [RmmPdev](#)
- [RmmRealm](#)

C2.15 RmmHipas type

The RmmHipas enumeration represents host IPA state.

The RmmHipas enumeration is an [abstract type](#).

The values of the RmmHipas enumeration are shown in the following table.

Name	Description
HIPAS_ASSIGNED	Protected IPA which is associated with a DATA Granule.
HIPAS_ASSIGNED_DEV	Protected IPA which is associated with a DEV_MAPPED Granule.
HIPAS_ASSIGNED_NS	Unprotected IPA which is associated with a physical Granule.
HIPAS_ASSIGNED_VSMMU	Protected IPA which is associated with a VSMMU Granule.
HIPAS_UNASSIGNED	Protected IPA which is not associated with any Granule.
HIPAS_UNASSIGNED_NS	Unprotected IPA which is not associated with any Granule.

C2.16 RmmLfaPolicy type

The RmmLfaPolicy enumeration represents a Live Firmware Activation policy.

The RmmLfaPolicy enumeration is an [abstract type](#).

The values of the RmmLfaPolicy enumeration are shown in the following table.

Name	Description
LFA_ALLOW	LFA is permitted.
LFA_DISALLOW	LFA is not permitted.

The RmmLfaPolicy enumeration is used in the following types:

- [RmmRealm](#)

C2.17 RmmMeasurementDescriptorData type

The RmmMeasurementDescriptorData structure contains data structure used to calculate the contribution to the RIM of a DATA Granule.

The RmmMeasurementDescriptorData structure is a [concrete type](#).

The width of the RmmMeasurementDescriptorData structure is 256 (0x100) bytes.

See also:

- [B4.3.1.4 RMI_DATA_CREATE extension of RIM](#)

The members of the RmmMeasurementDescriptorData structure are shown in the following table.

Name	Byte offset	Type	Description
desc_type	0x0	Bits8	Measurement descriptor type, value 0x0
len	0x8	UInt64	Length of this data structure in bytes
rim	0x10	RmmRealmMeasurement	Current RIM value
ipa	0x50	Address	IPA at which the DATA Granule is mapped in the Realm
flags	0x58	RmmDataFlags	Flags provided by Host

Name	Byte offset	Type	Description
content	0x60	RmmRealmMeasurement	Hash of contents of DATA Granule, or zero if flags indicate DATA Granule contents are unmeasured

Unused bits of the RmmMeasurementDescriptorData structure MBZ.

C2.18 RmmMeasurementDescriptorRec type

The RmmMeasurementDescriptorRec structure contains data structure used to calculate the contribution to the RIM of a REC.

The RmmMeasurementDescriptorRec structure is a [concrete type](#).

The width of the RmmMeasurementDescriptorRec structure is 256 (0x100) bytes.

See also:

- [B4.3.28.4 RMI_REC_CREATE extension of RIM](#)

The members of the RmmMeasurementDescriptorRec structure are shown in the following table.

Name	Byte offset	Type	Description
desc_type	0x0	Bits8	Measurement descriptor type, value 0x1
len	0x8	UInt64	Length of this data structure in bytes
rim	0x10	RmmRealmMeasurement	Current RIM value
content	0x50	RmmRealmMeasurement	Hash of 4KB page which contains REC parameters data structure

Unused bits of the RmmMeasurementDescriptorRec structure MBZ.

C2.19 RmmMeasurementDescriptorRipas type

The RmmMeasurementDescriptorRipas structure contains data structure used to calculate the contribution to the RIM of a RIPAS change.

The RmmMeasurementDescriptorRipas structure is a [concrete type](#).

The width of the RmmMeasurementDescriptorRipas structure is 256 (0x100) bytes.

See also:

- [B4.3.42.4 RMI_RTT_INIT_RIPAS extension of RIM](#)

The members of the RmmMeasurementDescriptorRipas structure are shown in the following table.

Name	Byte offset	Type	Description
desc_type	0x0	Bits8	Measurement descriptor type, value 0x2
len	0x8	UInt64	Length of this data structure in bytes

Name	Byte offset	Type	Description
rim	0x10	RmmRealmMeasurement	Current RIM value
base	0x50	Address	Base IPA of the RIPAS change
top	0x58	Address	Top IPA of the RIPAS change

Unused bits of the RmmMeasurementDescriptorRipas structure MBZ.

C2.20 RmmMecPolicy type

The RmmMecPolicy enumeration represents a MEC policy.

The RmmMecPolicy enumeration is an [abstract type](#).

The values of the RmmMecPolicy enumeration are shown in the following table.

Name	Description
MEC_POLICY_PRIVATE	The MEC protects memory owned by a single Realm. A MEC with this policy may be referred to as a <i>Private MEC</i> .
MEC_POLICY_SHARED	The MEC protects memory owned by multiple Realms. A MEC with this policy may be referred to as a <i>Shared MEC</i> .

The RmmMecPolicy enumeration is used in the following types:

- [RmmRealm](#)

C2.21 RmmMecState type

The RmmMecState enumeration represents state of a MEC.

The RmmMecState enumeration is an [abstract type](#).

The values of the RmmMecState enumeration are shown in the following table.

Name	Description
MEC_STATE_PRIVATE_ASSIGNED	A Private MEC which is assigned to a Realm.
MEC_STATE_PRIVATE_UNASSIGNED	A Private MEC which is not assigned to a Realm.
MEC_STATE_SHARED	A Shared MEC.

C2.22 RmmMemPermLocked type

The RmmMemPermLocked enumeration represents whether a memory permission value is locked.

The RmmMemPermLocked enumeration is an [abstract type](#).

The values of the RmmMemPermLocked enumeration are shown in the following table.

Name	Description
MEM_PERM_LOCKED	Memory permission value is locked
MEM_PERM_UNLOCKED	Memory permission value is unlocked

The RmmMemPermLocked enumeration is used in the following types:

- [RmmRealm](#)

C2.23 RmmMemPerms type

The RmmMemPerms structure contains memory permissions.

The RmmMemPerms structure is an [abstract type](#).

The members of the RmmMemPerms structure are shown in the following table.

Name	Type	Description
values	Bits64 [16]	Mapping from memory permission index to memory permission label Values use architectural encodings.

The RmmMemPerms structure is used in the following types:

- [RmmRealm](#)

C2.24 RmmPdev type

The RmmPdev structure contains attributes of a PDEV.

The RmmPdev structure is an [abstract type](#).

The members of the RmmPdev structure are shown in the following table.

Name	Type	Description
pdev_id	Bits64	Device identifier
prot	RmmPdevProtection	Configuration of protection between system and device
segment_id	Bits8	Segment identifier PCIe Segment identifier of the Root Port and endpoint.
ecam_addr	Address	ECAM base address of the PCIe configuration space.
root_id	Bits16	Root Port identifier Physical PCIe routing identifier of the Root Port to which the endpoint is connected.
cert_id	UInt64	Certificate identifier
rid_base	Bits16	Base of requester ID range (inclusive). The value is in PCI BDF format.

Name	Type	Description
rid_top	Bits16	Top of requester ID range (exclusive). The value is in PCI BDF format.
hash_algo	RmmHashAlgorithm	Algorithm used to generate device digests
ide_sid	UInt64	IDE stream ID
ncoh_num_addr_range	UInt64	Number of device non-coherent address ranges
ncoh_addr_range	RmmAddressRange[16]	Device non-coherent address range
coh_num_addr_range	UInt64	Number of device coherent address ranges
coh_addr_range	RmmAddressRange[4]	Device coherent address range
aux	Address[32]	Addresses of auxiliary Granules
num_aux	UInt64	Number of auxiliary Granules
state	RmmPdevState	Lifecycle state
comm_state	RmmDevCommState	Device communication state
num_vdevs	UInt64	Number of VDEVs associated with this PDEV

C2.25 RmmPdevProtection type

The RmmPdevProtection enumeration represents configuration of protection between system and device.

The RmmPdevProtection enumeration is an [abstract type](#).

See also:

- [A9.1.1 Device protection types](#)

The values of the RmmPdevProtection enumeration are shown in the following table.

Name	Description
PDEV_FCOH_E2E_IDE	Fully-coherent device with end-to-end protection provided by IDE.
PDEV_FCOH_E2E_SYS	Fully-coherent device with end-to-end protection provided by system construction.
PDEV_IOCOH_E2E_IDE	IO-coherent device with end-to-end protection provided by IDE.
PDEV_IOCOH_E2E_SYS	IO-coherent device with end-to-end protection provided by system construction.

The RmmPdevProtection enumeration is used in the following types:

- [RmmPdev](#)

C2.26 RmmPdevState type

The RmmPdevState enumeration represents the state of a PDEV.

The RmmPdevState enumeration is an [abstract type](#).

The values of the RmmPdevState enumeration are shown in the following table.

Name	Description
PDEV_COMMUNICATING	The RMM is communicating with the device.
PDEV_ERROR	Device has reported a fatal error.
PDEV_HAS_KEY	RMM has device public key.
PDEV_IDE_RESETTING	The PDEV's IDE link is being reset.
PDEV_NEEDS_KEY	RMM needs device public key.
PDEV_NEW	Initial state of the device.
PDEV_READY	Secure connection between the RMM and the device has been established. Physical link between the device and memory is secured. Ready for creation of VDEV instances.
PDEV_STOPPED	Secure connection between the RMM and the device has been terminated.
PDEV_STOPPING	The RMM is communicating with the device to terminate the secure connection between the RMM and the device.

The RmmPdevState enumeration is used in the following types:

- [RmmPdev](#)

C2.27 RmmPhysicalAddressSpace type

The RmmPhysicalAddressSpace enumeration represents the PAS of a Granule.

The RmmPhysicalAddressSpace enumeration is an [abstract type](#).

See also:

- [B3.30 GranuleAccessPermitted function](#)

The values of the RmmPhysicalAddressSpace enumeration are shown in the following table.

Name	Description
PAS_NS	Non-secure PAS.
PAS_REALM	Realm PAS.
PAS_ROOT	Root PAS.
PAS_SECURE	Secure PAS.

C2.28 RmmRdev type

The RmmRdev structure contains attributes of an RDEV.

The RmmRdev structure is an [abstract type](#).

The members of the RmmRdev structure are shown in the following table.

Name	Type	Description
state	RmmRdevState	Lifecycle state
operation	RmmRdevOperation	Operation being performed by the RDEV
vdev_ptr	Address	PA of VDEV associated with this RDEV

C2.29 RmmRdevOperation type

The RmmRdevOperation enumeration represents operation being performed by an RDEV.

The RmmRdevOperation enumeration is an [abstract type](#).

The values of the RmmRdevOperation enumeration are shown in the following table.

Name	Description
RDEV_OP_GET_INTERFACE_REPORT	RDEV is handling an RSI_RDEV_GET_INTERFACE_REPORT request.
RDEV_OP_GET_MEASUREMENTS	RDEV is handling an RSI_RDEV_GET_MEASUREMENTS request.
RDEV_OP_LOCK	RDEV is handling an RSI_RDEV_LOCK request.
RDEV_OP_NONE	No operation is being performed.
RDEV_OP_START	RDEV is handling an RSI_RDEV_START request.

The RmmRdevOperation enumeration is used in the following types:

- [RmmRdev](#)

C2.30 RmmRdevState type

The RmmRdevState enumeration represents the state of an RDEV.

The RmmRdevState enumeration is an [abstract type](#).

The values of the RmmRdevState enumeration are shown in the following table.

Name	Description
RDEV_ERROR	Device interface has reported a fatal error.
RDEV_LOCKED	Device interface is locked.
RDEV_LOCKED_BUSY	Device interface is locked and is handling an interruptible Realm device operation.
RDEV_STARTED	Device interface is started.
RDEV_STARTED_BUSY	Device interface is started and is handling an interruptible Realm device operation.
RDEV_STOPPED	Device interface is stopped.

Name	Description
RDEV_STOPPING	Device interface is stopping.
RDEV_UNLOCKED	Device interface is unlocked.
RDEV_UNLOCKED_BUSY	Device interface is unlocked and is handling an interruptible Realm device operation.

The RmmRdevState enumeration is used in the following types:

- [RmmRdev](#)

C2.31 RmmRealm type

The RmmRealm structure contains attributes of a Realm.

The RmmRealm structure is an [abstract type](#).

See also:

- [A2.1 Realm](#)

The members of the RmmRealm structure are shown in the following table.

Name	Type	Description
feat_lpa2	RmmFeature	Whether LPA2 is enabled for this Realm
ipa_width	UInt8	IPA width in bits
measurements	RmmRealmMeasurement [5]	Realm measurements
hash_algo	RmmHashAlgorithm	Algorithm used to compute Realm measurements
rec_index	UInt64	Index of next REC to be created
rtt_base	Address [4]	Realm Translation Table base addresses If rtt_tree_per_plane is FEATURE_FALSE then only the first entry is valid. If rtt_tree_per_plane is FEATURE_TRUE then only the first (num_aux_planes + 1) entries are valid.
rtt_level_start	Int64	RTT starting level
rtt_num_start	UInt64	Number of physically contiguous starting level RTTs
state	RmmRealmState	Lifecycle state
vmid	Bits16 [4]	Virtual Machine Identifiers If rtt_tree_per_plane is FEATURE_FALSE then only the first entry is valid. If rtt_tree_per_plane is FEATURE_TRUE then only the first (num_aux_planes + 1) entries are valid.
rpv	Bits512	Realm Personalization Value
feat_da	RmmFeature	Whether Realm device assignment is enabled for this Realm
rtt_tree_per_plane	RmmFeature	Whether this Realm has an RTT tree per Plane

Name	Type	Description
num_aux_planes	UInt64	Number of auxiliary Planes
rtt_s2ap_encoding	RmmRttS2APEncoding	S2AP encoding
overlay_perms	RmmMemPerms[4]	Memory overlay permissions
overlay_locked	RmmMemPermLocked[16]	Whether memory overlay value is locked
lfa_policy	RmmLfaPolicy	Live Firmware Activation policy for components within the Realm's TCB
mecid	Bits64	Memory Encryption Context Identifier
mec_policy	RmmMecPolicy	MEC policy
num_recs	UInt64	Number of RECs owned by this Realm
num_vdevs	UInt64	Number of VDEVs owned by this Realm
vdev_count	UInt64	Number of VDEVs which have been assigned to this Realm

C2.32 RmmRealmMeasurement type

The RmmRealmMeasurement type is realm measurement.

The RmmRealmMeasurement type is a [concrete type](#).

The width of the RmmRealmMeasurement type is 512 bits.

C2.33 RmmRealmState type

The RmmRealmState enumeration represents the state of a Realm.

The RmmRealmState enumeration is an [abstract type](#).

The values of the RmmRealmState enumeration are shown in the following table.

Name	Description
REALM_ACTIVE	Eligible for execution.
REALM_NEW	Under construction. Not eligible for execution.
REALM_SYSTEM_OFF	System has been turned off. Not eligible for execution.

The RmmRealmState enumeration is used in the following types:

- [RmmRealm](#)

C2.34 RmmRec type

The RmmRec structure contains attributes of a REC.

The RmmRec structure is an [abstract type](#).

See also:

- [A2.3 Realm Execution Context](#)

The members of the RmmRec structure are shown in the following table.

Name	Type	Description
owner	Address	PA of RD of Realm which owns this REC
aux	Address [16]	PA of auxiliary Granules
flags	RmmRecFlags	Flags which control REC behavior
mpidr	Bits64	MPIDR value
gic_owner	UInt64	Index of Plane which is the GIC owner
state	RmmRecState	Lifecycle state
pending	RmmRecPending	Whether a REC operation is pending
emulatable_abort	RmmRecEmulatableAbort	Whether the most recent exit from this REC was due to an Emulatable Data Abort
gprs	Bits64 [32]	General-purpose register values
pc	Bits64	Program counter value
sysregs	RmmSystemRegisters	EL1 and EL0 system register values
attest_state	RmmRecAttestState	Attestation token generation state
attest_challenge	Bits512	Challenge for under-construction attestation token
ripas_addr	Address	Next IPA to be processed in RIPAS change
ripas_top	Address	Top IPA of pending RIPAS change
ripas_value	RmmRipas	RIPAS value of pending RIPAS change
ripas_destroyed	RmmRipasChangeDestroyed	Whether a RIPAS change from DESTROYED should be permitted
ripas_response	RmmRecResponse	Host response to RIPAS change request
dev_mem_addr	Address	Next IPA to be processed in device memory mapping validation
dev_mem_top	Address	Top IPA of pending device memory mapping validation
dev_mem_pa	Address	PA of device memory
dev_mem_flags	RmmDevMemFlags	Device memory mapping validation flags
dev_mem_response	RmmRecResponse	Host response to device memory mapping validation request
s2ap_addr	Address	Next IPA to be processed in S2AP change
s2ap_top	Address	Top IPA of pending S2AP change
s2ap_overlay_index	UInt4	Overlay index of pending S2AP change
s2ap_response	RmmRecResponse	Host response to S2AP change request
vdev_id	Bits64	Virtual device ID
inst_id	UInt64	Device instance ID
inst_id_valid	RmmBoolean	Whether device instance ID is valid

C2.35 RmmRecAttestState type

The RmmRecAttestState enumeration represents whether an attestation token generation operation is ongoing on this REC.

The RmmRecAttestState enumeration is an [abstract type](#).

The values of the RmmRecAttestState enumeration are shown in the following table.

Name	Description
ATTEST_IN_PROGRESS	An attestation token generation operation is in progress.
NO_ATTEST_IN_PROGRESS	No attestation token generation operation is in progress.

The RmmRecAttestState enumeration is used in the following types:

- [RmmRec](#)

C2.36 RmmRecEmulatableAbort type

The RmmRecEmulatableAbort enumeration represents whether the most recent exit from a REC was due to an Emulatable Data Abort.

The RmmRecEmulatableAbort enumeration is an [abstract type](#).

The values of the RmmRecEmulatableAbort enumeration are shown in the following table.

Name	Description
EMULATABLE_ABORT	The most recent exit from a REC was due to an Emulatable Data Abort.
NOT_EMULATABLE_ABORT	The most recent exit from a REC was not due to an Emulatable Data Abort.

The RmmRecEmulatableAbort enumeration is used in the following types:

- [RmmRec](#)

C2.37 RmmRecFlags type

The RmmRecFlags structure contains REC flags.

The RmmRecFlags structure is an [abstract type](#).

The members of the RmmRecFlags structure are shown in the following table.

Name	Type	Description
runnable	RmmRecRunnable	Whether the REC is eligible to run

The RmmRecFlags structure is used in the following types:

- [RmmRec](#)

C2.38 RmmRecPending type

The RmmRecPending enumeration represents whether a REC operation is pending.

The RmmRecPending enumeration is an [abstract type](#).

The values of the RmmRecPending enumeration are shown in the following table.

Name	Description
REC_PENDING_HOST_CALL	A Host call is pending.
REC_PENDING_NONE	No operation is pending.
REC_PENDING_PSCI	A PSCI operation is pending.
REC_PENDING_VDEV_REQUEST	A VDEV request is pending.

The RmmRecPending enumeration is used in the following types:

- [RmmRec](#)

C2.39 RmmRecResponse type

The RmmRecResponse enumeration represents whether the Host accepted or rejected a Realm request.

The RmmRecResponse enumeration is an [abstract type](#).

The values of the RmmRecResponse enumeration are shown in the following table.

Name	Description
ACCEPT	Host accepted the Realm request.
REJECT	Host rejected the Realm request.

The RmmRecResponse enumeration is used in the following types:

- [RmmRec](#)

C2.40 RmmRecRunnable type

The RmmRecRunnable enumeration represents whether a REC is eligible for execution.

The RmmRecRunnable enumeration is an [abstract type](#).

The values of the RmmRecRunnable enumeration are shown in the following table.

Name	Description
NOT_RUNNABLE	Not eligible for execution.
RUNNABLE	Eligible for execution.

The RmmRecRunnable enumeration is used in the following types:

- [RmmRecFlags](#)

C2.41 RmmRecState type

The RmmRecState enumeration represents the state of a REC.

The RmmRecState enumeration is an [abstract type](#).

The values of the RmmRecState enumeration are shown in the following table.

Name	Description
REC_READY	REC is not currently running.
REC_RUNNING	REC is currently running.

The RmmRecState enumeration is used in the following types:

- [RmmRec](#)

C2.42 RmmRipas type

The RmmRipas enumeration represents realm IPA state.

The RmmRipas enumeration is an [abstract type](#).

The values of the RmmRipas enumeration are shown in the following table.

Name	Description
DESTROYED	Address which is inaccessible to the Realm due to an action taken by the Host.
DEV	Address where memory of an assigned Realm device is mapped.
EMPTY	Address where no Realm resources are mapped.
RAM	Address where private code or data owned by the Realm is mapped.

The RmmRipas enumeration is used in the following types:

- [RmmRec](#)
- [RmmRttEntry](#)

C2.43 RmmRipasChangeDestroyed type

The RmmRipasChangeDestroyed enumeration represents whether a RIPAS change from DESTROYED should be permitted.

The RmmRipasChangeDestroyed enumeration is an [abstract type](#).

The values of the RmmRipasChangeDestroyed enumeration are shown in the following table.

Name	Description
CHANGE_DESTROYED	A RIPAS change from DESTROYED should be permitted.
NO_CHANGE_DESTROYED	A RIPAS change from DESTROYED should not be permitted.

The RmmRipasChangeDestroyed enumeration is used in the following types:

- [RmmRec](#)

C2.44 RmmRtt type

The RmmRtt structure contains an RTT.

The RmmRtt structure is an [abstract type](#).

The members of the RmmRtt structure are shown in the following table.

Name	Type	Description
entries	RmmRttEntry [512]	Entries

C2.45 RmmRttEntry type

The RmmRttEntry structure contains attributes of an RTT Entry.

The RmmRttEntry structure is an [abstract type](#).

See also:

- [A5.6 Realm Translation Table](#)

The members of the RmmRttEntry structure are shown in the following table.

Name	Type	Description
addr	Address	Output address
ripas	RmmRipas	RIPAS
state	RmmRttEntryState	State
attr_prot	RmmRttMemAttr	Memory type and cacheability attributes for a Protected IPA This attribute and attr_unprot are aliased views of the underlying MemAttr field in the RTT descriptor. This view is valid if the RTT entry describes an address in Protected IPA space. The RMM uses stage 2 memory attributes to constrain the resultant memory type and cacheability attributes, based on the type of physical location identified by the output address.

Name	Type	Description
attr_unprot	Bits3	Memory type and cacheability attributes for an Unprotected IPA This attribute and attr_prot are aliased views of the underlying MemAttr field in the RTT descriptor. This view is valid if the RTT entry describes an address in Unprotected IPA space. The Host controls memory type and cacheability attributes by setting the value of the MemAttr[2:0] field in the RTT descriptor.
sh	RmmRttShareability	Shareability attributes.
s2ap_direct	RmmRttS2APDirect	Directly-encoded S2AP This attribute is valid if the RTT entry describes an address in Unprotected IPA space and the Realm uses direct S2AP encoding.
s2ap_indirect	RmmRttS2APIndirect	Directly-encoded S2AP This attribute is valid if either of the following is true: <ul style="list-style-type: none"> • The RTT entry describes an address in Protected IPA space. • The RTT entry describes an address in Unprotected IPA space and the Realm uses indirect S2AP encoding.

The RmmRttEntry structure is used in the following types:

- [RmmRtt](#)
- [RmmRttWalkResult](#)

C2.46 RmmRttEntryState type

The RmmRttEntryState enumeration represents the state of an RTTE.

The RmmRttEntryState enumeration is an [abstract type](#).

The values of the RmmRttEntryState enumeration are shown in the following table.

Name	Description
ASSIGNED	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a DATA Granule.
ASSIGNED_DEV	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a DEV_MAPPED Granule.
ASSIGNED_NS	This RTTE is identified by an Unprotected IPA. The output address of this RTTE points to a Granule-aligned address within NS PAS.
ASSIGNED_VSMMU	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a VSMMU Granule.

Name	Description
AUX_DESTROYED	An auxiliary RTT was destroyed while a corresponding primary RTT entry was live.
TABLE	The output address of this RTTE points to the next-level RTT.
UNASSIGNED	This RTTE is identified by a Protected IPA. This RTTE is not associated with any Granule.
UNASSIGNED_NS	This RTTE is identified by an Unprotected IPA. This RTTE is not associated with any Granule.

The RmmRttEntryState enumeration is used in the following types:

- [RmmRttEntry](#)

C2.47 RmmRttMemAttr type

The RmmRttMemAttr enumeration represents memory type and cacheability attributes.

The RmmRttMemAttr enumeration is an [abstract type](#).

The values of the RmmRttMemAttr enumeration are shown in the following table.

Name	Description
MEMATTR_CACHEABLE	Memory type and cacheability attributes for a mapping to a cacheable location.
MEMATTR_NON_CACHEABLE	Memory type and cacheability attributes for a mapping to a non-cacheable location.
MEMATTR_PASSTHROUGH	Pass through memory type and cacheability attributes from stage 1 translation.

The RmmRttMemAttr enumeration is used in the following types:

- [RmmRttEntry](#)

C2.48 RmmRttPlaneFeature type

The RmmRttPlaneFeature enumeration represents RTT usage models supported for multi-Plane Realms.

The RmmRttPlaneFeature enumeration is an [abstract type](#).

See also:

- [A3.11 Support for auxiliary Planes](#)

The values of the RmmRttPlaneFeature enumeration are shown in the following table.

Name	Description
RTT_PLANE_AUX	A multi-Plane Realm uses auxiliary RTTs

Name	Description
RTT_PLANE_AUX_SINGLE	A multi-Plane Realm can be configured to either use auxiliary RTTs, or a single RTT
RTT_PLANE_SINGLE	A multi-Plane Realm uses a single RTT

The RmmRttPlaneFeature enumeration is used in the following types:

- [RmmFeatures](#)

C2.49 RmmRttProtected type

The RmmRttProtected enumeration represents specifies whether an RTT entry is in Protected IPA space or Unprotected IPA space.

The RmmRttProtected enumeration is an [abstract type](#).

The values of the RmmRttProtected enumeration are shown in the following table.

Name	Description
RTT_PROTECTED	Protected IPA space.
RTT_UNPROTECTED	Unprotected IPA space.

C2.50 RmmRttS2APDirect type

The RmmRttS2APDirect structure contains directly-encoded S2AP.

The RmmRttS2APDirect structure is an [abstract type](#).

The members of the RmmRttS2APDirect structure are shown in the following table.

Name	Type	Description
read	RmmBoolean	Read permission
write	RmmBoolean	Write permission

The RmmRttS2APDirect structure is used in the following types:

- [RmmRttEntry](#)

C2.51 RmmRttS2APEncoding type

The RmmRttS2APEncoding enumeration represents encoding used for S2AP.

The RmmRttS2APEncoding enumeration is an [abstract type](#).

The values of the RmmRttS2APEncoding enumeration are shown in the following table.

Name	Description
S2AP_DIRECT	Direct S2AP encoding.
S2AP_INDIRECT	Indirect S2AP encoding.

The RmmRttS2APEncoding enumeration is used in the following types:

- [RmmRealm](#)

C2.52 RmmRttS2APIndirect type

The RmmRttS2APIndirect structure contains indirectly-encoded S2AP.

The RmmRttS2APIndirect structure is an [abstract type](#).

The members of the RmmRttS2APIndirect structure are shown in the following table.

Name	Type	Description
base_index	UInt4	Base permission index
overlay_index	UInt4	Overlay permission index

The RmmRttS2APIndirect structure is used in the following types:

- [RmmRttEntry](#)

C2.53 RmmRttShareability type

The RmmRttShareability enumeration represents shareability attributes.

The RmmRttShareability enumeration is an [abstract type](#).

The values of the RmmRttShareability enumeration are shown in the following table.

Name	Description
SHAREABILITY_INNER	Inner Shareable.
SHAREABILITY_OUTER	Outer Shareable.

The RmmRttShareability enumeration is used in the following types:

- [RmmRttEntry](#)

C2.54 RmmRttWalkNotAligned type

The RmmRttWalkNotAligned structure contains result of an RTT walk which is not aligned to the requested level.

The RmmRttWalkNotAligned structure is an [abstract type](#).

The members of the RmmRttWalkNotAligned structure are shown in the following table.

Name	Type	Description
valid	RmmBoolean	TRUE if an RTT walk was performed whose result is not aligned to the requested level
index	UInt64	RTT index
addr	Address	Address
walk	RmmRttWalkResult	Walk result

C2.55 RmmRttWalkResult type

The RmmRttWalkResult structure contains result of an RTT walk.

The RmmRttWalkResult structure is an [abstract type](#).

See also:

- [A5.6.10 RTT walk](#)

The members of the RmmRttWalkResult structure are shown in the following table.

Name	Type	Description
level	Int8	RTT level reached by the walk
rtt_addr	Address	Address of RTT reached by the walk
rtte	RmmRttEntry	RTTE reached by the walk

The RmmRttWalkResult structure is used in the following types:

- [RmmRttWalkNotAligned](#)

C2.56 RmmSystemRegisters type

The RmmSystemRegisters structure contains EL0 and EL1 system registers.

The RmmSystemRegisters structure is an [abstract type](#).

The RmmSystemRegisters structure is used in the following types:

- [RmmRec](#)

C2.57 RmmVdev type

The RmmVdev structure contains attributes of a VDEV.

The RmmVdev structure is an [abstract type](#).

The members of the RmmVdev structure are shown in the following table.

Name	Type	Description
vdev_id	Bits64	Virtual device identifier

Name	Type	Description
tdi_id	Bits64	TDI identifier
inst_id	UInt64	Instance identifier
pdev	Address	PA of parent PDEV
realm	Address	PA of RD of Realm which owns this VDEV
state	RmmVdevState	Lifecycle state
comm_state	RmmDevCommState	Device communication state
aux	Address[32]	Addresses of auxiliary Granules
num_aux	UInt64	Number of auxiliary Granules
vsmmu	RmmFeature	Whether device uses a VSMMU
vsmmu_addr	Address	PA of VSMMU. This field is valid if vsmmu is FEATURE_TRUE.
vsid	Bits64	Virtual Stream Identifier. This field is valid if vsmmu is FEATURE_TRUE.

C2.58 RmmVdevState type

The RmmVdevState enumeration represents the state of a VDEV.

The RmmVdevState enumeration is an [abstract type](#).

The values of the RmmVdevState enumeration are shown in the following table.

Name	Description
VDEV_COMMUNICATING	The RMM is communicating with the VDEV.
VDEV_ERROR	Device interface has reported a fatal error.
VDEV_NEW	Initial state of the device.
VDEV_READY	No device transaction is associated with the VDEV.
VDEV_STOPPED	Device interface is stopped.
VDEV_STOPPING	The RMM is communicating with the VDEV to stop the device interface.

The RmmVdevState enumeration is used in the following types:

- [RmmVdev](#)

C2.59 RmmVsmmu type

The RmmVsmmu structure contains attributes of a VSMMU.

The RmmVsmmu structure is an [abstract type](#).

The members of the RmmVsmmu structure are shown in the following table.

Name	Type	Description
realm	Address	PA of RD of Realm which owns this VSMMU
reg_base	Address	Base IPA of register base in Realm's Protected IPA space
reg_top	Address	Top IPA of register base in Realm's Protected IPA space
aidr	Bits64	SMMU_AIDR register value
idr	Bits64[7]	SMMU_IDR register values

DRAFT

Chapter C3

Generic types

This section defines types which are shared between RMM interfaces and descriptions of RMM abstract state.

See also:

- [B4.4 RMI types](#)
- [B5.4 RSI types](#)
- [B6.4 PSCI types](#)
- [Chapter C2 RMM types](#)

C3.1 Address type

The Address type is an address.

The Address type is a [concrete type](#).

The width of the Address type is 64 bits.

C3.2 BitsN type

The BitsN type is an N-bit field.

The BitsN type is a [concrete type](#).

The width of the BitsN type is N bits.

C3.3 IntN type

The IntN type is an signed N-bit integer.

The IntN type is a [concrete type](#).

The width of the IntN type is N bits.

C3.4 UIntN type

The UIntN type is an unsigned N-bit integer.

The UIntN type is a [concrete type](#).

The width of the UIntN type is N bits.

DRAFT

DRAFT

Part D
Usage

Chapter D1

Flows

This section presents flows which explain how the RMM architecture can be used by the Host, and by Realm software.

Note that parts of the sequences below are for illustration only. For example, in the Realm creation flows, the `RMI_GRANULE_DELEGATE` and `RMI_GRANULE_UNDELEGATE` commands are called immediately before or after the `RMI_X_CREATE` and `RMI_X_DESTROY` commands respectively. An alternative flow would be for the Host to maintain a pool of Granules in the `DELEGATED` state, from which RMM data structures and Realm data can be allocated on demand.

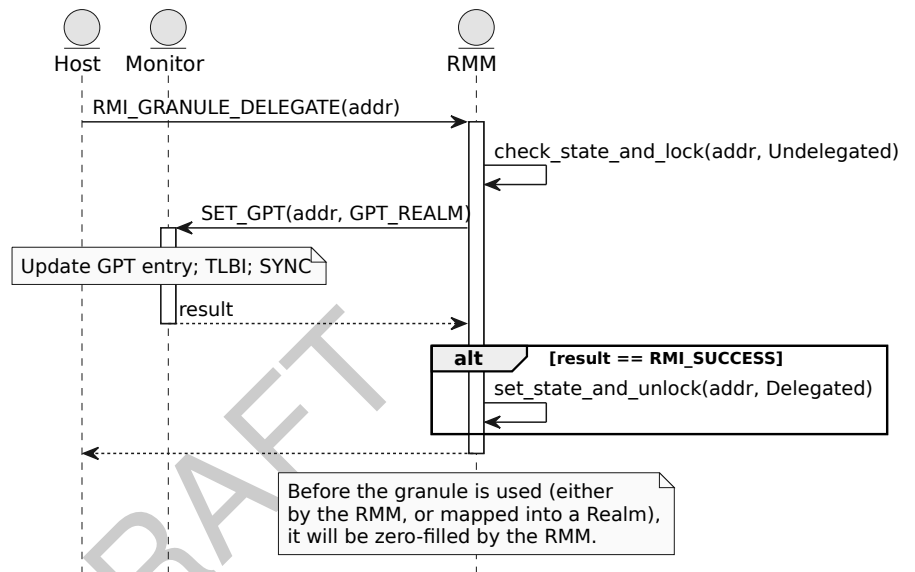
D1.1 Granule delegation flows

D1.1.1 Granule delegation flow

The following diagram shows how the GPT entry of a Granule is changed to GPT_REALM.

See [Arm Architecture Reference Manual Supplement, The Realm Management Extension \(RME\), for Armv9-A \[2\]](#) for example software flows for the operations performed by the Monitor in this flow.

It is anticipated that the Monitor software will be required to use synchronization mechanisms to serialize access to the GPT.



See also:

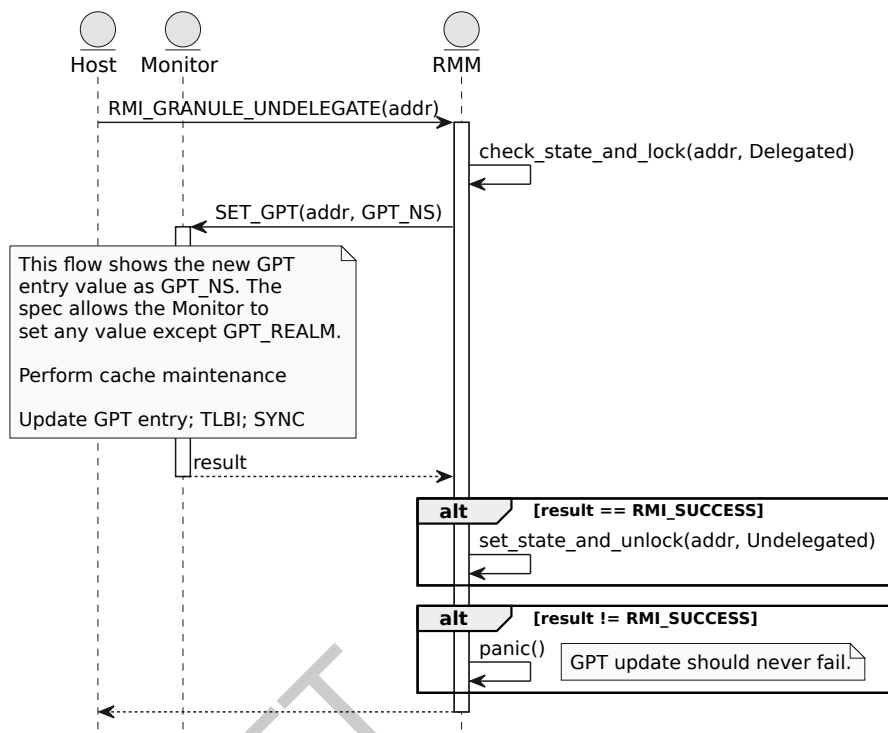
- [A2.2.2 Granule attributes](#)
- [B4.3.7 RMI_GRANULE_DELEGATE command](#)
- [D1.1.2 Granule undelegation flow](#)

D1.1.2 Granule undelegation flow

The following diagram shows how the GPT entry of a Granule is changed from GPT_REALM.

See [Arm Architecture Reference Manual Supplement, The Realm Management Extension \(RME\), for Armv9-A \[2\]](#) for example software flows for the operations performed by the Monitor in this flow.

It is anticipated that the Monitor software will be required to use synchronization mechanisms to serialize access to the GPT.



See also:

- [A2.2.2 Granule attributes](#)
- [B4.3.8 RMI_GRANULE_UNDELEGATE command](#)
- [D1.1.1 Granule delegation flow](#)

D1.2 Realm lifecycle flows

This section contains flows which relate to the Realm lifecycle.

See also:

- [A2.1.5 Realm lifecycle](#)

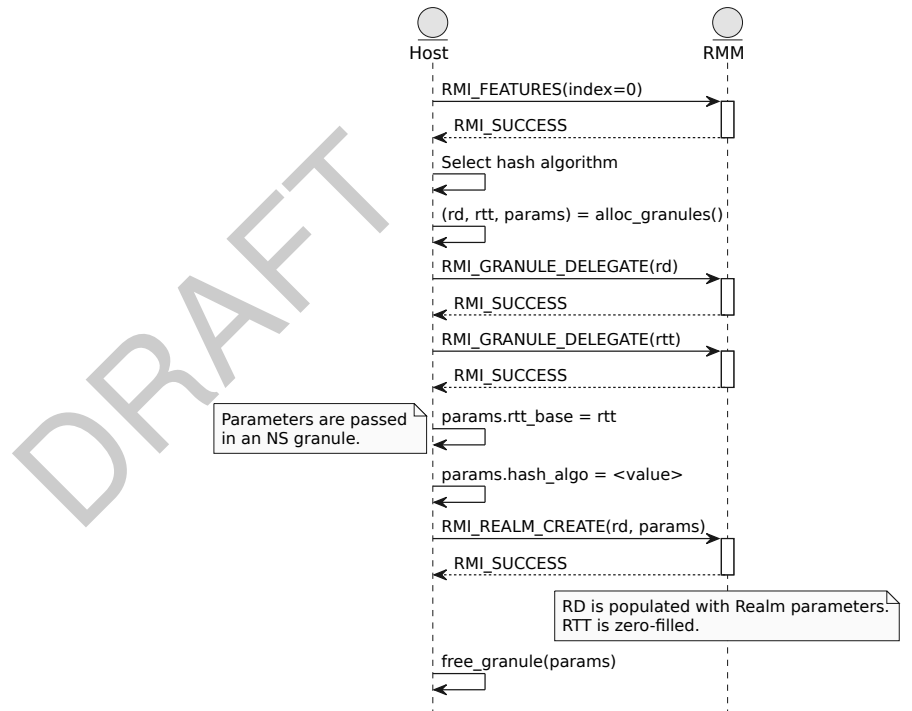
D1.2.1 Realm creation flow

The following diagram shows the flow for creating a Realm.

To create a Realm, the Host must allocate and delegate two Granules:

- `rd` to store the Realm Descriptor
- `rtt` which will be the starting level Realm Translation Table (RTT)

The Host also provides an NS Granule (`params`) containing Realm creation parameters.



See also:

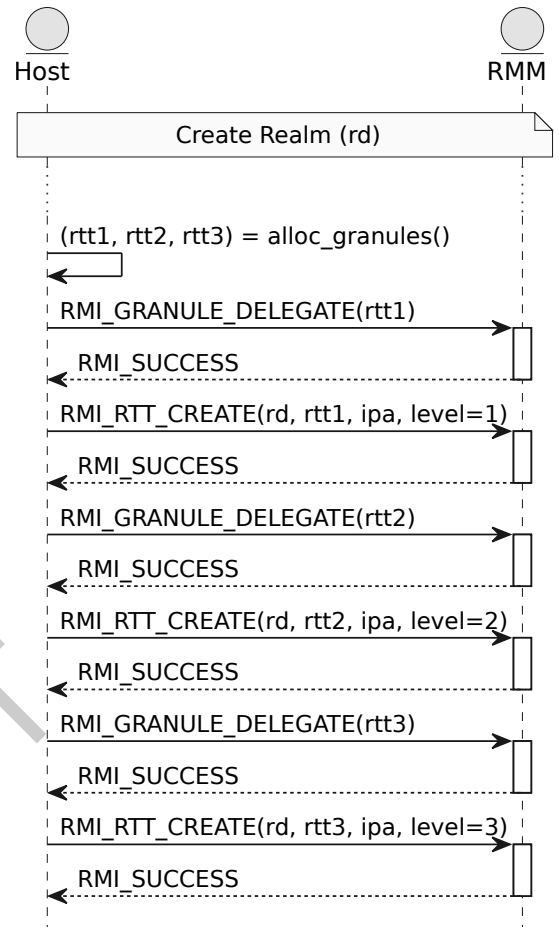
- [B4.3.7 RMI_GRANULE_DELEGATE command](#)
- [B4.3.25 RMI_REALM_CREATE command](#)
- [D1.2.5 Realm destruction flow](#)

D1.2.2 Realm Translation Table creation flow

The following diagram shows the flow for populating the Realm Translation Tables (RTTs).

The starting level Realm Translation Tables (RTTs) are provided at Realm creation time.

Subsequent levels of RTT are added using the `RMI_RTT_CREATE` command.



See also:

- [Chapter A5 Realm memory management](#)
- [B4.3.38 RMI_RTT_CREATE command](#)
- [D1.2.1 Realm creation flow](#)
- [D1.2.3 Initialize memory of New Realm flow](#)

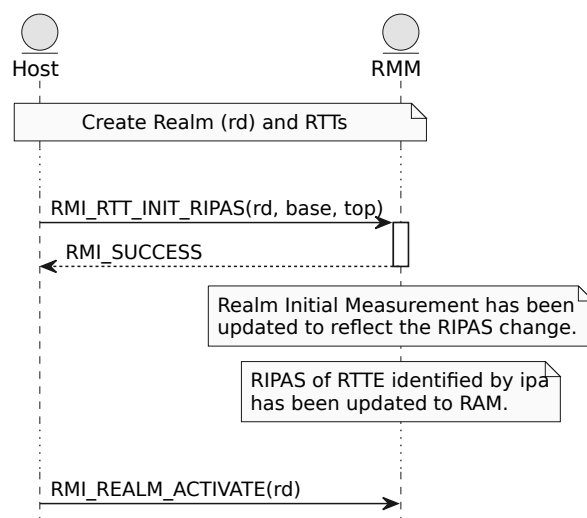
D1.2.3 Initialize memory of New Realm flow

Immediately following Realm creation, every page in the Protected IPA space has its RIPAS set to EMPTY. There are two ways in which the Host can set the RIPAS of a given page of Protected IPA space to RAM:

1. Change the RIPAS by executing RMI_RTT_INIT_RIPAS, but do not populate the contents of the page. The RIM is extended to reflect the RIPAS change.
2. Both change the RIPAS and populate the page with contents provided by the Host, by executing RMI_DATA_CREATE. The RIM is extended to reflect the contents added by the Host.

Once the Host has performed either of these actions for a given page of Protected IPA space, that page cannot be further modified prior to Realm activation.

The following diagram shows the flow for initializing the RIPAS without providing contents.

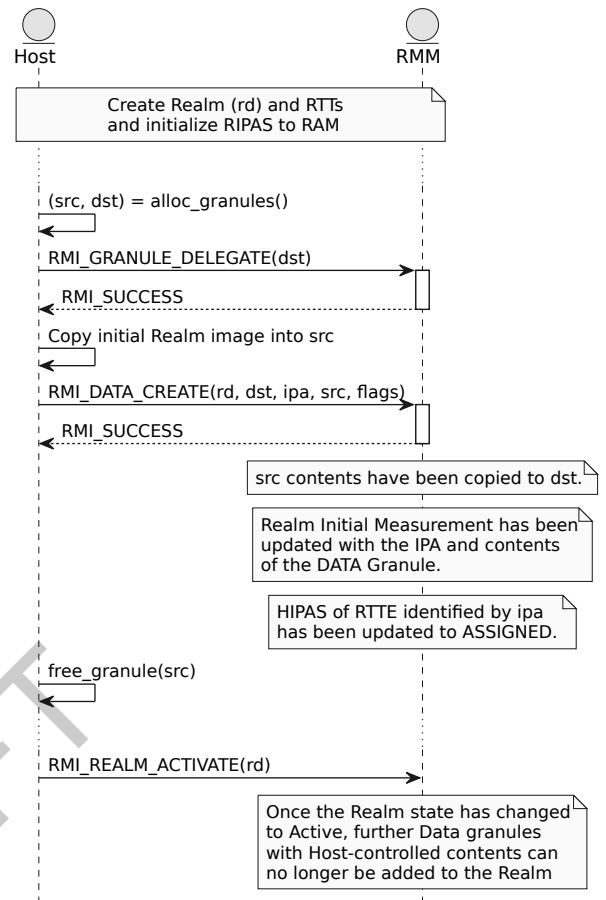


The following diagram shows the flow for populating the page with contents provided by the Host.

To do this, the Host must:

- Delegate a destination Granule (*dst*).
- Provide an NS Granule (*src*), whose contents will be copied into the destination Granule.
- Specify the Protected IPA *ipa* at which the *dst* Granule should be mapped in the Realm's IPA space.
- Ensure that the level 3 RTT which contains the RTTE identified by the Protected IPA has been created.

Once the Data Granule has been created, the *src* Granule can be reallocated by the Host.



See also:

- [A2.2.2 Granule attributes](#)
- [A5.2.2 Realm IPA state](#)
- [A7.1.1 Realm Initial Measurement](#)
- [B4.3.1 RMI_DATA_CREATE command](#)
- [B4.3.7 RMI_GRANULE_DELEGATE command](#)
- [B4.3.42 RMI_RTT_INIT_RIPAS command](#)
- [D1.2.1 Realm creation flow](#)
- [D1.2.2 Realm Translation Table creation flow](#)
- [D1.2.5 Realm destruction flow](#)

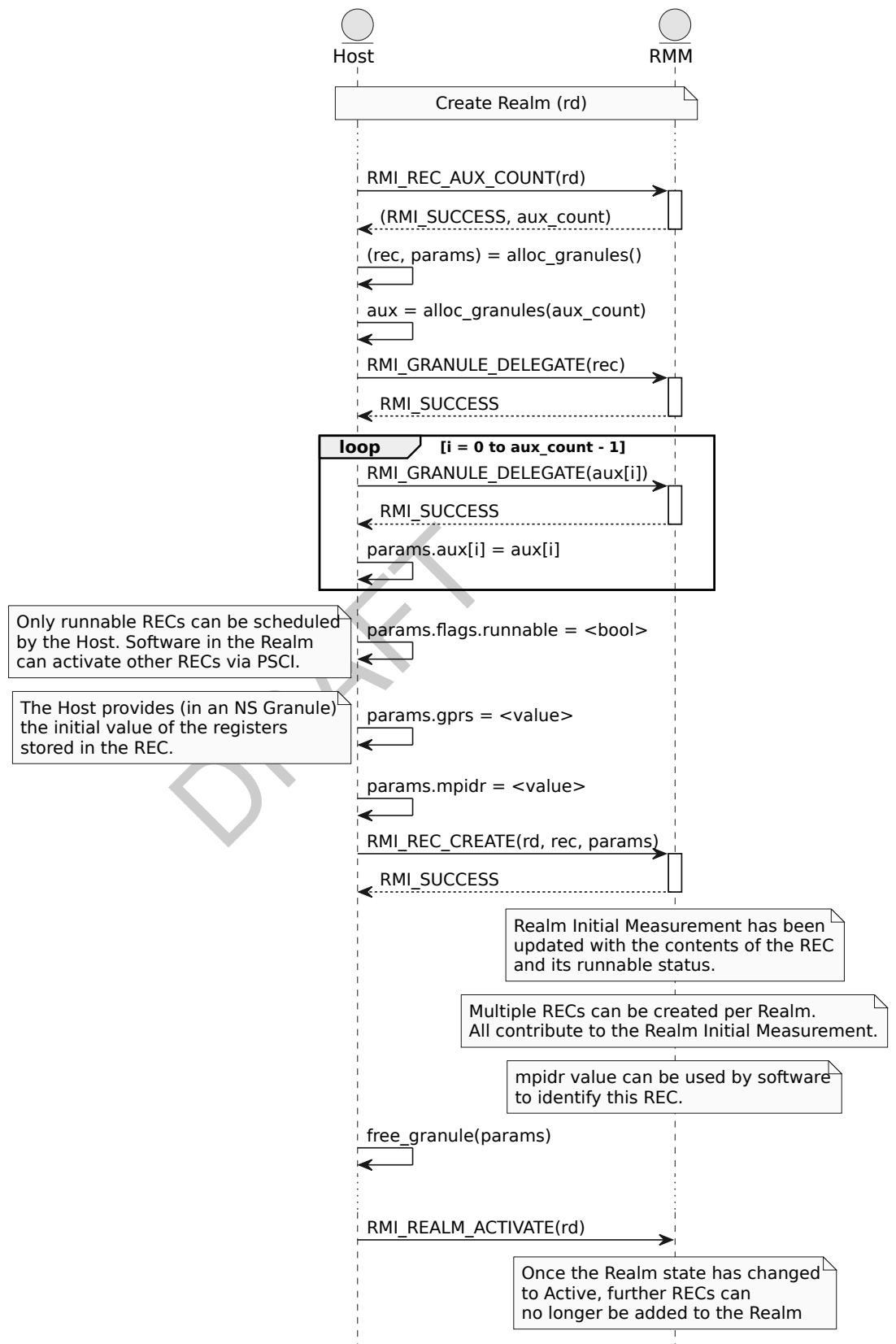
D1.2.4 REC creation flow

The following diagram shows the flow for creating a REC during Realm creation.

To create a REC, the Host must:

- Delegate a destination Granule (*rec*).
- Query the number of auxiliary Granules required, by calling `RMI_REC_AUX_COUNT`
- Delegate the required number of auxiliary Granules (*aux*)
- Provide auxiliary Granule addresses, register values and REC activation status in an NS Granule (*params*).

Once the REC has been created, the *params* Granule can be reallocated by the Host.



See also:

- [B4.3.7 RMI_GRANULE_DELEGATE command](#)
- [B4.3.27 RMI_REC_AUX_COUNT command](#)
- [B4.3.28 RMI_REC_CREATE command](#)
- [D1.2.1 Realm creation flow](#)
- [D1.2.5 Realm destruction flow](#)

D1.2.5 Realm destruction flow

The following diagram shows the flow for destroying a Realm.

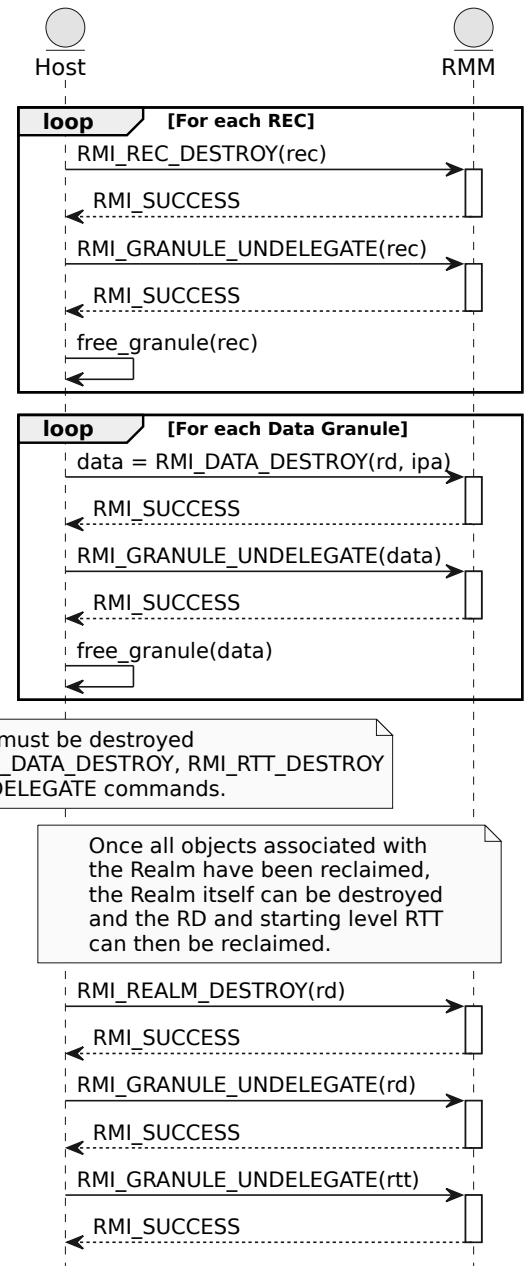
To destroy a Realm, the Host must first make the Realm non-live. This is done by destroying (in any order) the objects which are associated with the Realm:

- Data Granules
- RECs
- RTTs

Finally, the Realm itself can be destroyed.

Once each of these objects has been destroyed, the corresponding Granules can be undelegated and reallocated by the Host.

DRAFT



See also:

- [A2.1.4 Realm liveness](#)
- [B4.3.3 RMI_DATA_DESTROY command](#)
- [B4.3.8 RMI_GRANULE_UNDELEGATE command](#)
- [B4.3.26 RMI_REALM_DESTROY command](#)
- [B4.3.29 RMI_REC_DESTROY command](#)
- [D1.2.1 Realm creation flow](#)

D1.3 Realm exception model flows

This section contains flows which relate to the Realm exception model.

See also:

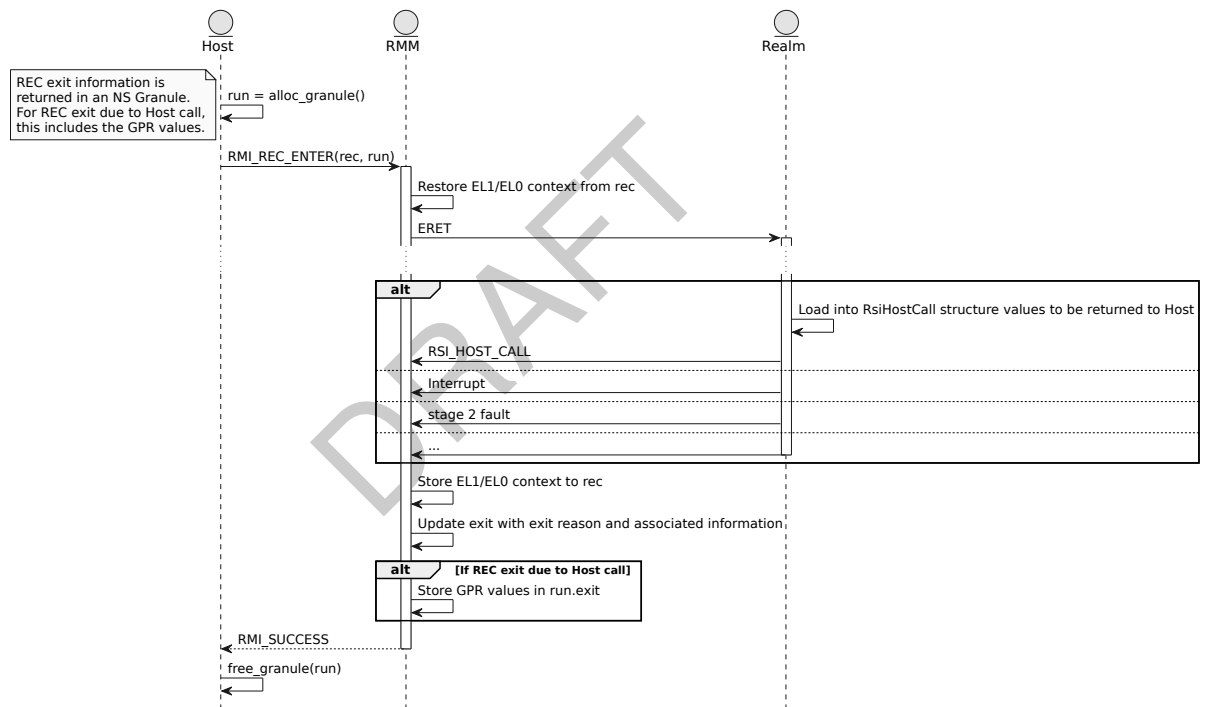
- [Chapter A4 Realm exception model](#)

D1.3.1 Realm entry and exit flow

The following diagram shows how a Realm is executed, and illustrates the different reasons for exiting the Realm and returning control to the Host.

A REC is entered using the RMI_REC_ENTER command. The parameters to this command include:

- an *RmiRecEnter* object, which is a data structure used to pass values from the Host to the RMM on REC entry
- an *RmiRecExit* object, which is a data structure used to pass values from the RMM to the Host on REC exit



See also:

- [Chapter A4 Realm exception model](#)
- [D1.3.2 Host call flow](#)
- [D1.3.3 REC exit due to Data Abort fault flow](#)
- [D1.3.4 MMIO emulation flow](#)

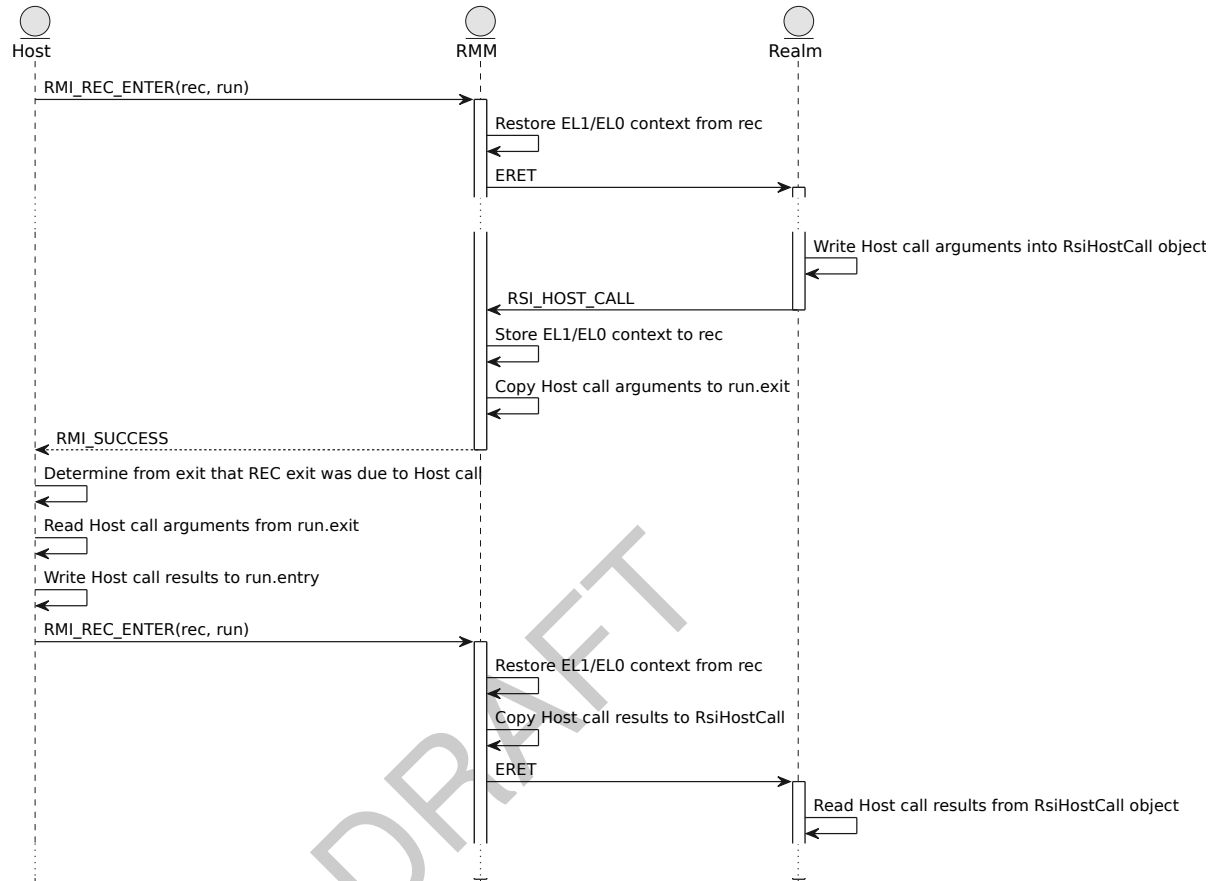
D1.3.2 Host call flow

The following diagram shows how software executing inside the Realm can voluntarily yield control back to the Host by making a Host call.

A REC is entered using the RMI_REC_ENTER command. The parameters to this command include:

- an *RmiRecEnter* object, which is a data structure used to pass values from the Host to the RMM on REC entry
- an *RmiRecExit* object, which is a data structure used to pass values from the RMM to the Host on REC exit

On execution of RSI_HOST_CALL, arguments are copied from the RsiHostCall object in Realm memory into the RmiRecExit object in NS memory. On the subsequent RMI_REC_ENTER, return values are copied from the RmiRecEnter object in NS memory into the RsiHostCall object in Realm memory.



See also:

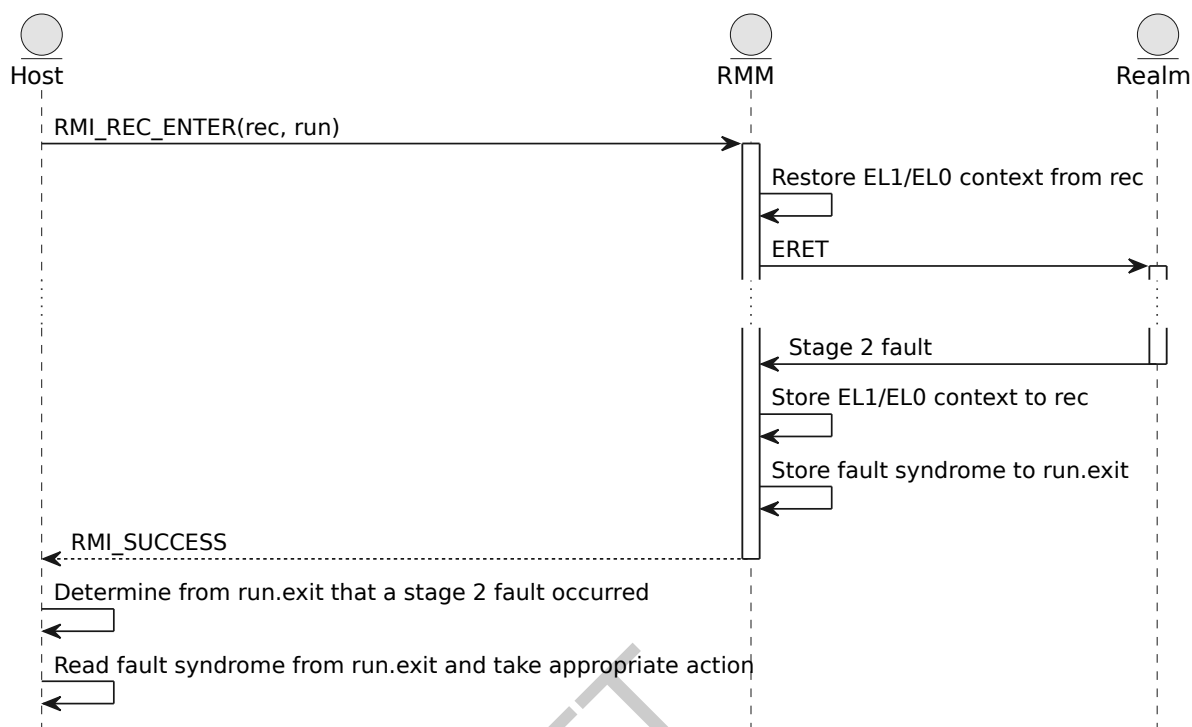
- [A4.5 Host call](#)

D1.3.3 REC exit due to Data Abort fault flow

The following diagram shows how a Data Abort due to a Realm access is taken to the Host.

A REC is entered using the RMI_REC_ENTER command. The parameters to this command include:

- an *RmiRecEnter* object, which is a data structure used to pass values from the Host to the RMM on REC entry
- an *RmiRecExit* object, which is a data structure used to pass values from the RMM to the Host on REC exit

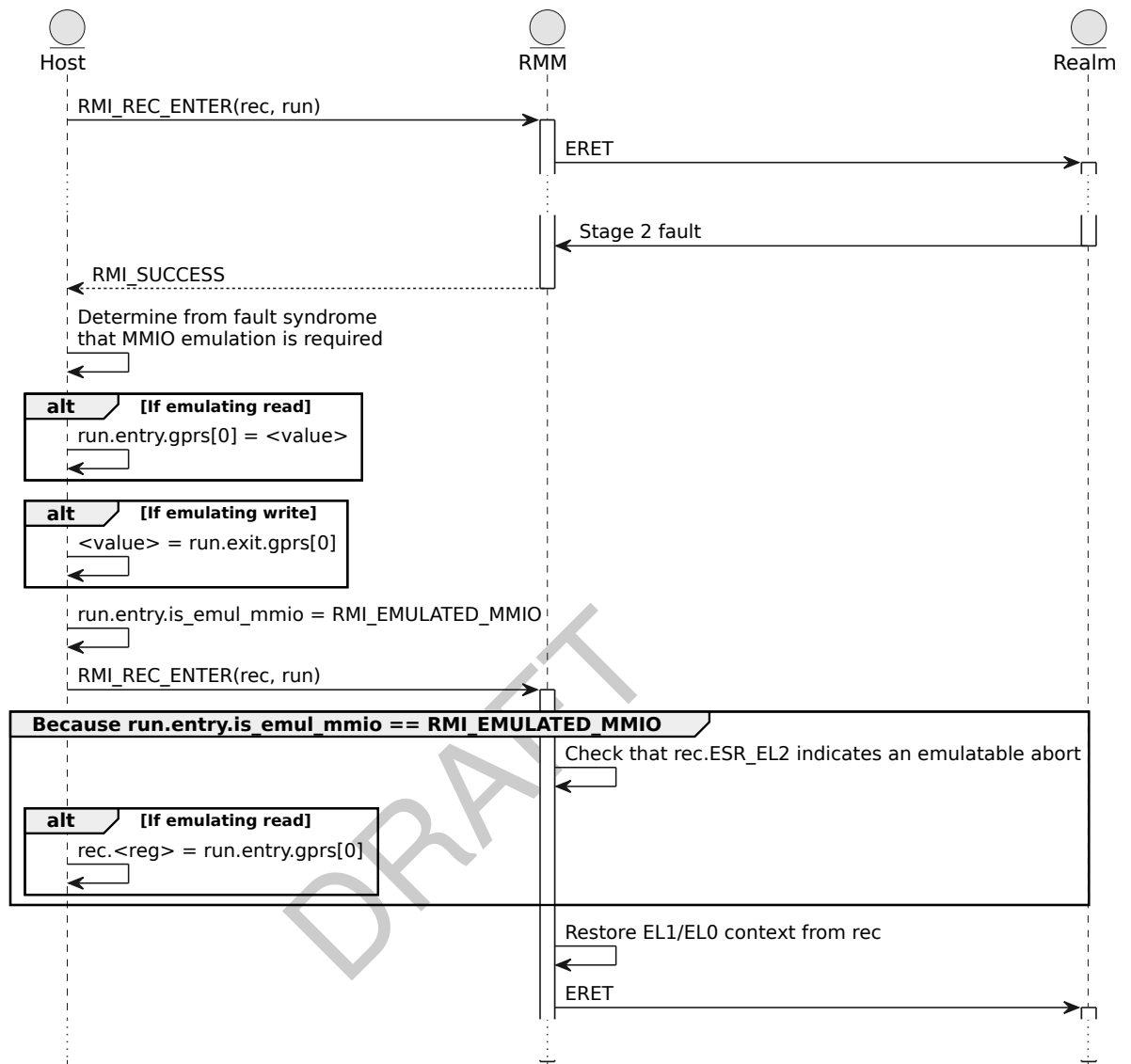


See also:

- [Chapter A4 Realm exception model](#)

D1.3.4 MMIO emulation flow

The following diagram shows how an MMIO access by a Realm can be emulated by the Host.



See also:

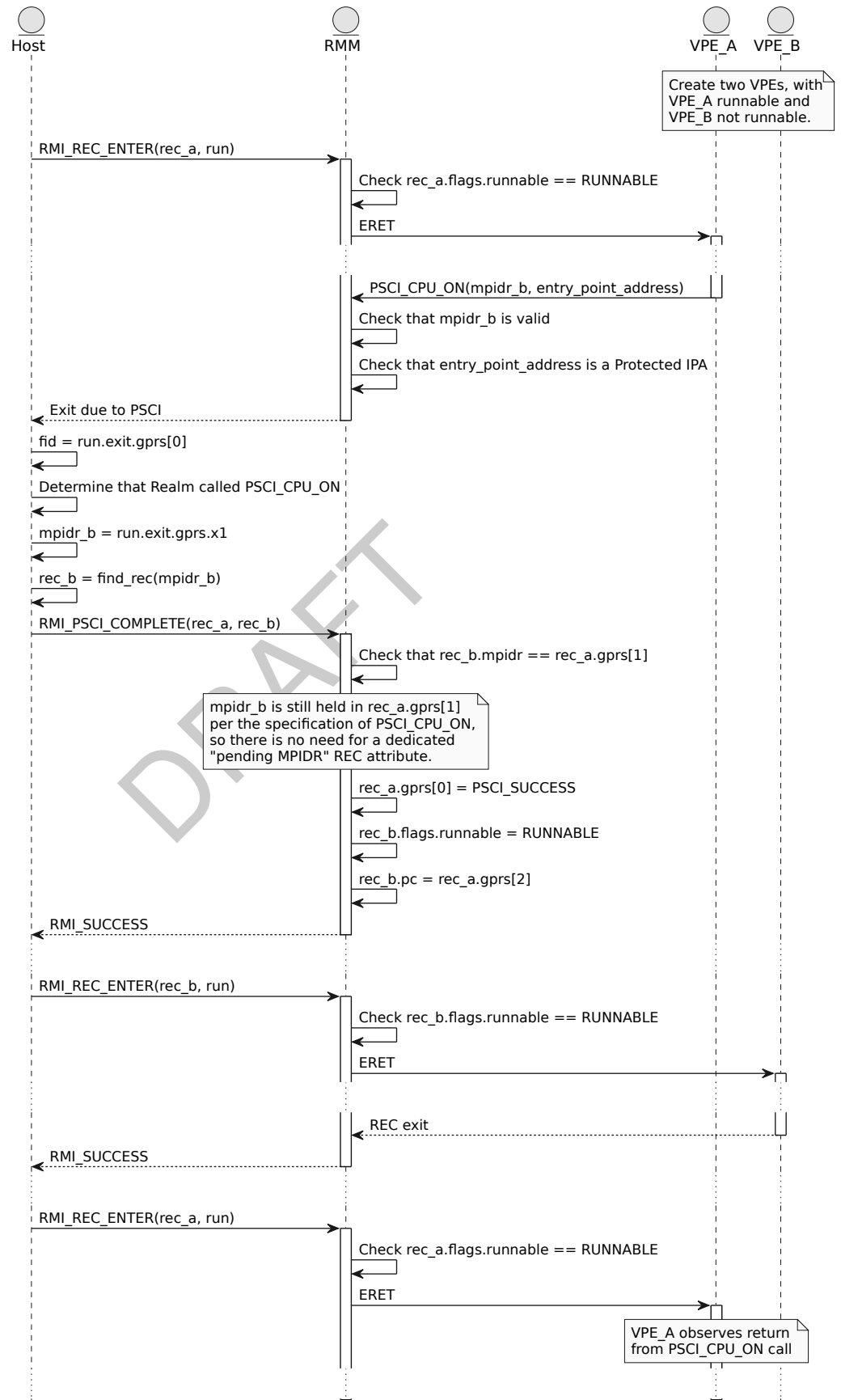
- [Chapter A4 Realm exception model](#)

D1.4 PSCI flows

D1.4.1 PSCI_CPU_ON flow

The following diagram shows how one Realm VPE can set the “runnable” flag in another Realm VPE by executing PSCI_CPU_ON.

DRAFT



See also:

- [B4.3.21 RMI_PSCI_COMPLETE command](#)
- [B6.3.3 PSCI_CPU_ON command](#)

DRAFT

D1.5 Realm memory management flows

This section contains flows which relate to management of Realm memory.

See also:

- [Chapter A5 Realm memory management](#)

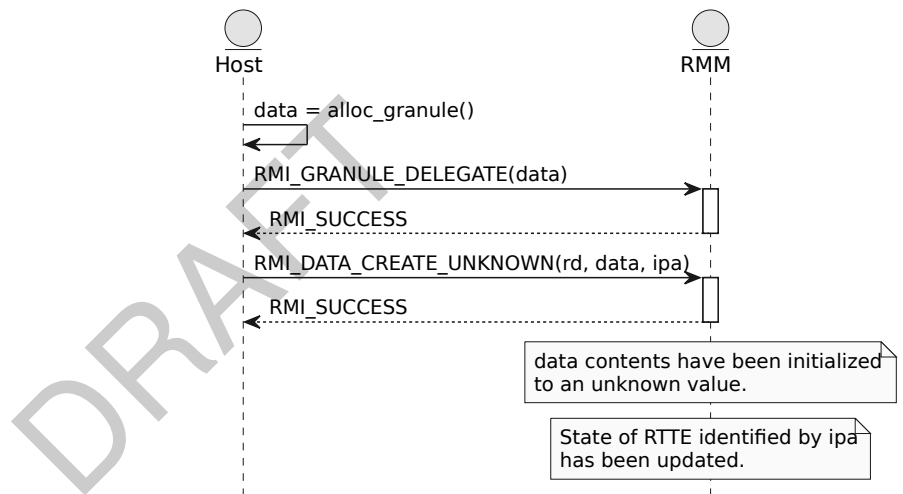
D1.5.1 Add memory to Active Realm flow

The following diagram shows the flow for adding memory to a Realm whose state is `REALM_ACTIVE`.

To add memory to a Realm whose state is `REALM_ACTIVE`, the Host must:

- Delegate a destination Granule (`dst`).
- Specify the Protected IPA at which the `dst` Granule will be mapped in the Realm's IPA space.
- Ensure that the level 3 RTT which contains the RTTE identified by the Protected IPA has been created.

Once a given Protected IPA has been populated with unknown content, it cannot be repopulated.

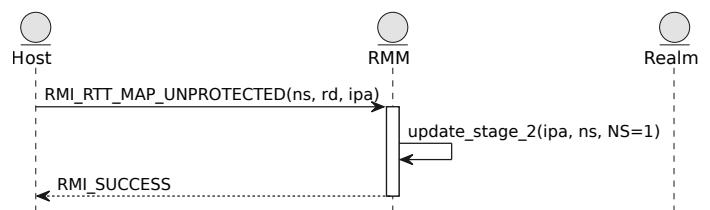


See also:

- [A2.1.5 Realm lifecycle](#)
- [Chapter A5 Realm memory management](#)
- [B4.3.2 RMI_DATA_CREATE_UNKNOWN command](#)
- [B4.3.7 RMI_GRANULE_DELEGATE command](#)

D1.5.2 NS memory flow

The following diagram describes how NS memory can be mapped into a Realm.



See also:

- [Chapter A5 Realm memory management](#)
- [B4.3.43 RMI_RTT_MAP_UNPROTECTED command](#)

- [B4.3.47 RMI_RTT_UNMAP_UNPROTECTED command](#)

D1.5.3 RIPAS change flow

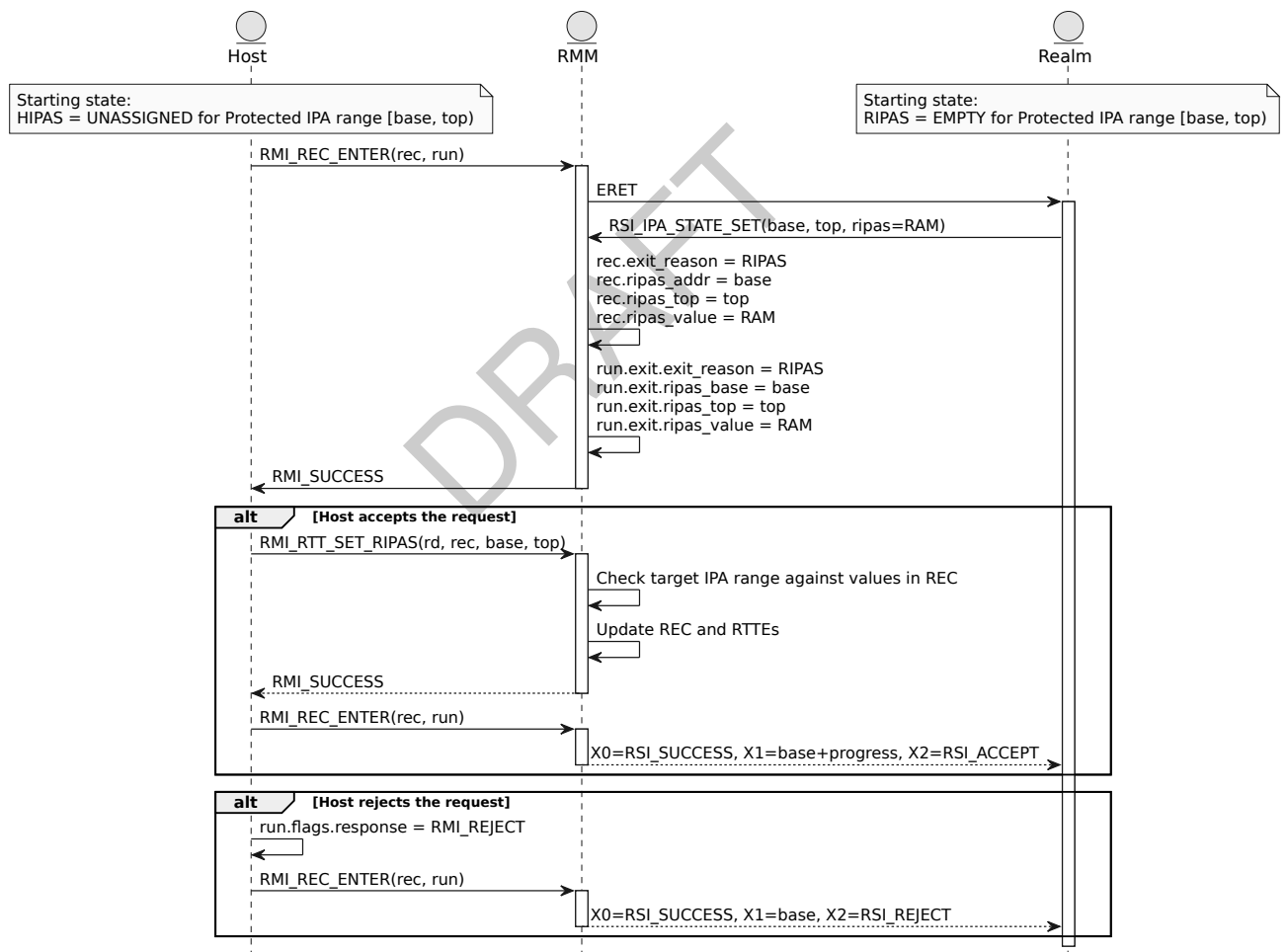
The following diagram describes how a Realm requests a RIPAS change, and how that request is handled by the Host.

- The Realm calls `RSI_IPA_STATE_SET` to request a RIPAS change for IPA range `[base, top)`.
- This causes a REC exit due to RIPAS change pending.

On taking a REC exit due to RIPAS change pending, the Host does the following:

- Reads the region base and top addresses from the `RmiRecExit` object.
- Applies the requested RIPAS change to an IPA range starting from the base of the target region, and extending no further than the top of the target region.
- Calls `RMI_REC_ENTER` to re-enter the REC.

The Realm observes in `X1` the top of the region for which the RIPAS change was applied.



See also:

- [A5.4 RIPAS change](#)
- [B4.3.30 RMI_REC_ENTER command](#)
- [B4.3.45 RMI_RTT_SET_RIPAS command](#)
- [B5.3.6 RSI_IPA_STATE_SET command](#)
- [D2.2 Realm shared memory protocol flow](#)

D1.5.4 S2AP change flow

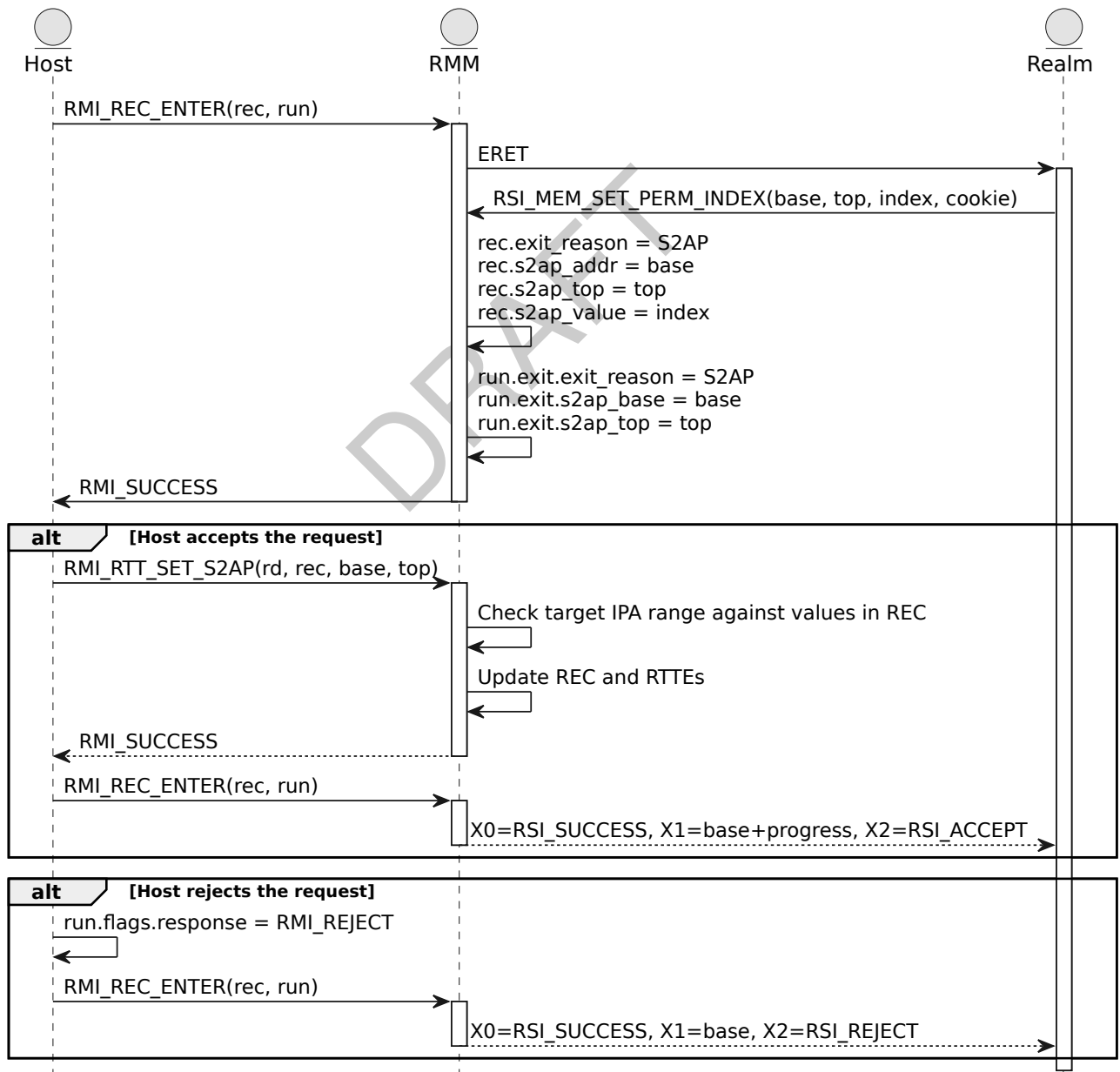
The following diagram describes how a Realm requests a S2AP change, and how that request is handled by the Host.

- The Realm calls RSI_MEM_SET_PERM_INDEX to request an S2AP change for IPA range [base, top).
- This causes a REC exit due to S2AP change pending.

On taking a REC exit due to S2AP change pending, the Host does the following:

- Reads the region base and top addresses from the RmiRecExit object.
- Applies the requested S2AP change to an IPA range starting from the base of the target region, and extending no further than the top of the target region.
- Calls RMI_REC_ENTER to re-enter the REC.

The Realm observes in X1 the top of the region for which the S2AP change was applied.



See also:

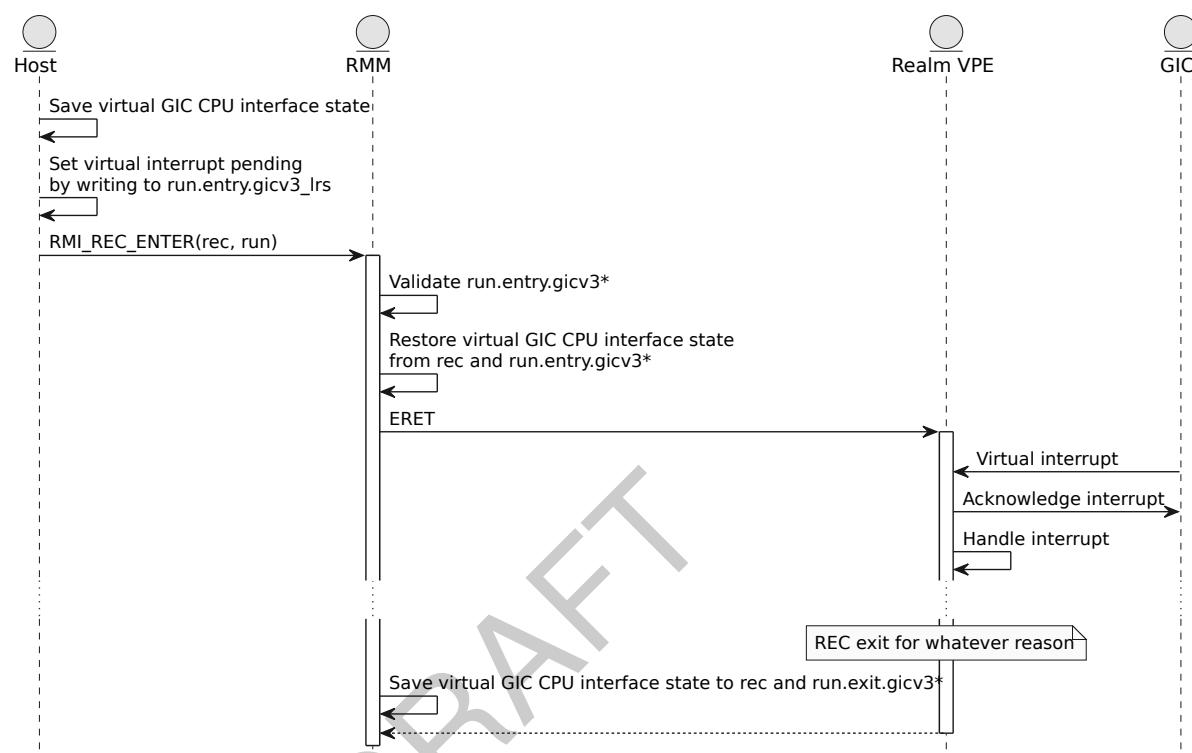
- [A10.3.3.2 Stage 2 Access Permissions change within a multi-Plane Realm](#)
- [B4.3.30 RMI_REC_ENTER command](#)
- [B4.3.46 RMI_RTT_SET_S2AP command](#)
- [B5.3.10 RSI_MEM_SET_PERM_INDEX command](#)

DRAFT

D1.6 Realm interrupts and timers flows

D1.6.1 Interrupt flow

The following diagram shows how a virtual interrupt is injected into a Realm by the Host.

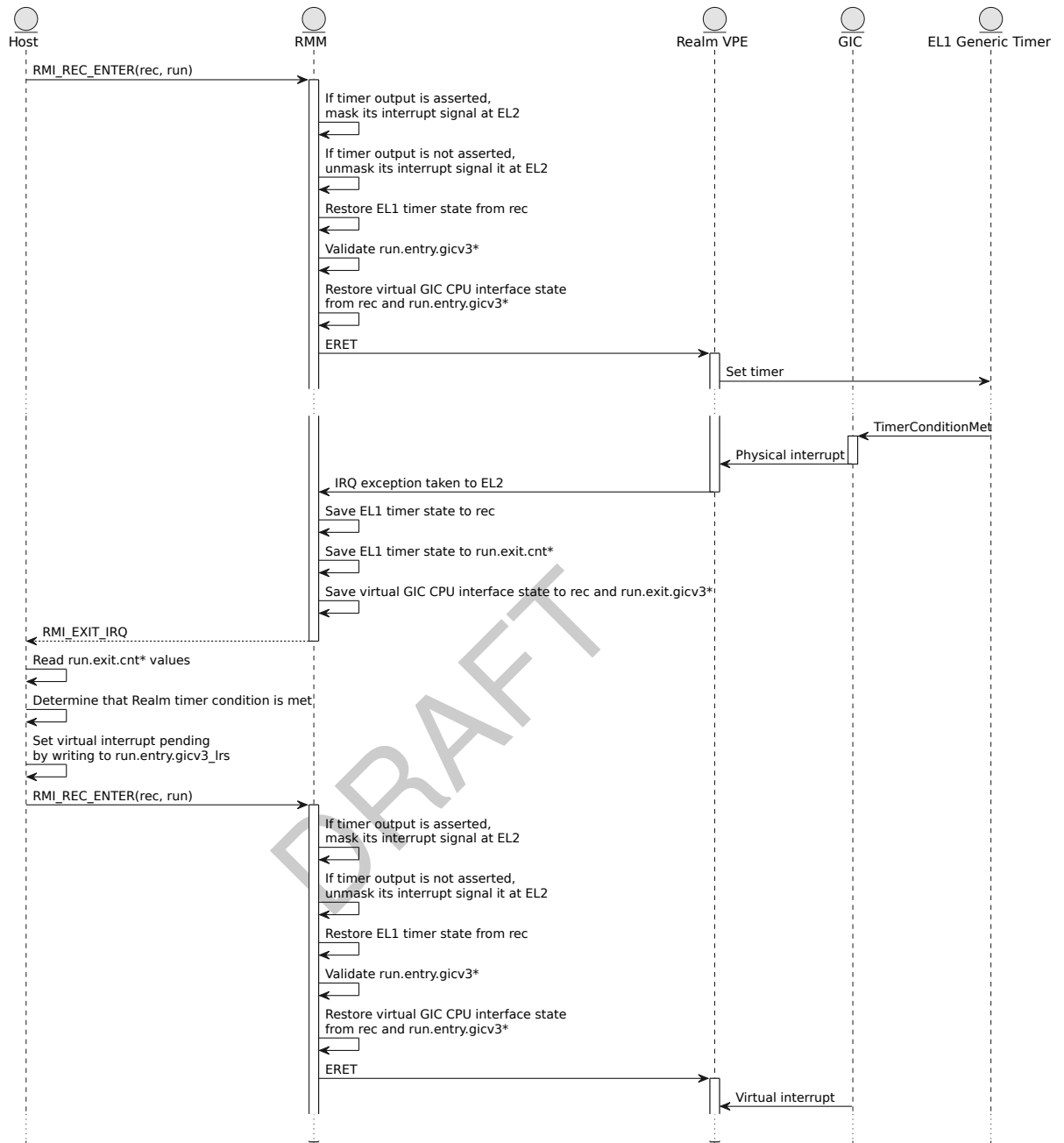


See also:

- [A6.1 Realm interrupts](#)

D1.6.2 Timer interrupt delivery flow

The following diagram shows how a timer interrupt is delivered to and handled by a Realm.



See also:

- [A6.2 Realm timers](#)

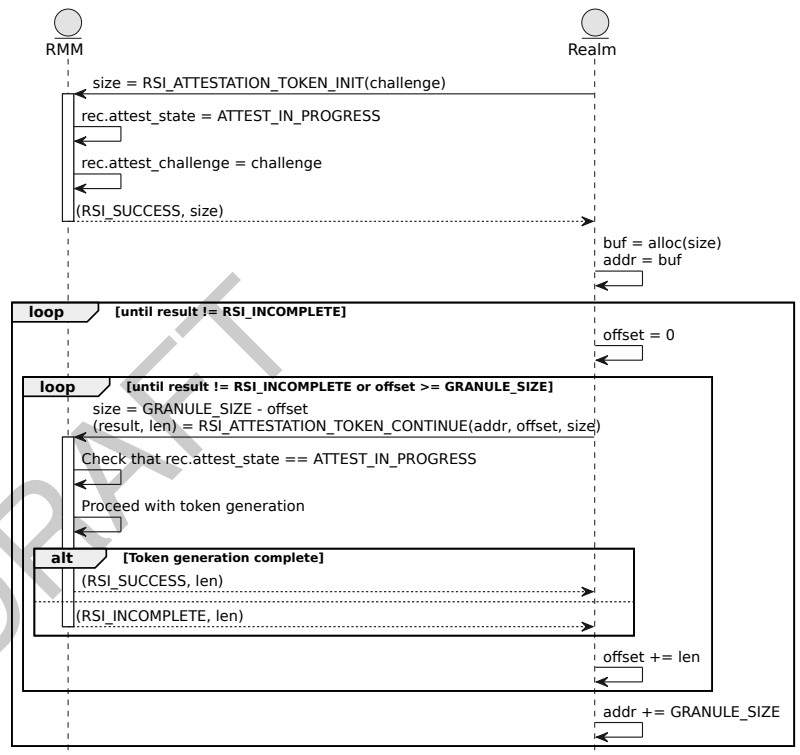
D1.7 Realm attestation flows

D1.7.1 Attestation token generation flow

The following diagram shows the flow for a Realm to obtain an attestation token.

The Realm first calls `RSI_ATTESTATION_TOKEN_INIT`, providing a challenge value. The output values include an upper bound on the attestation token size.

The Realm then calls `RSI_ATTESTATION_TOKEN_CONTINUE`, providing the address of a buffer where the next part of the attestation token will be written. This command is called in a loop, until the result is not `RSI_INCOMPLETE`.



See also:

- [A7.2.2 Attestation token generation](#)
- [B5.3.1 RSI_ATTESTATION_TOKEN_CONTINUE command](#)
- [B5.3.2 RSI_ATTESTATION_TOKEN_INIT command](#)

D1.7.2 Handling interrupts during attestation token generation flow

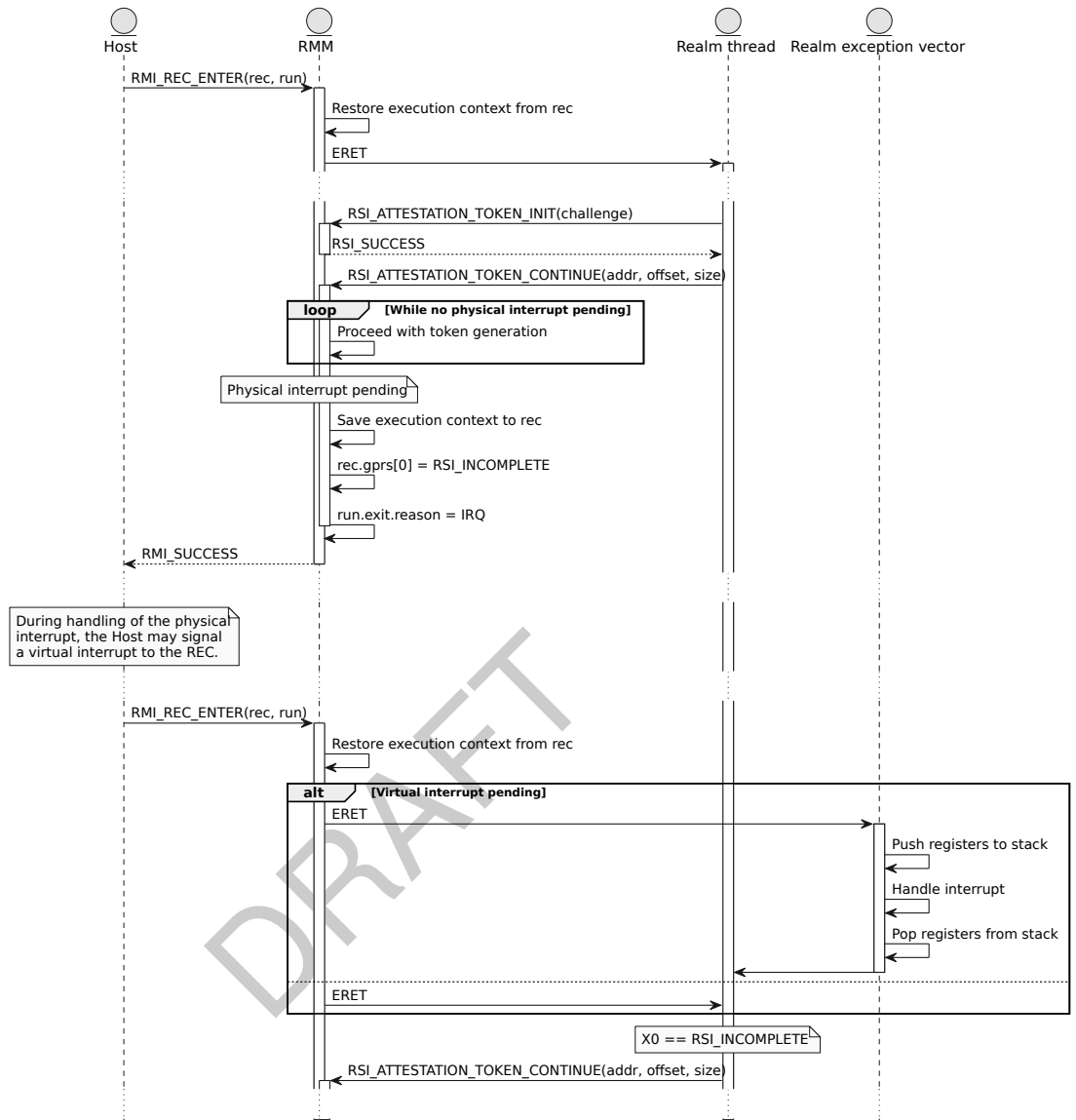
The following diagram shows how interrupts are handled during generation of an attestation token.

If the RMM detects that a physical interrupt is pending during execution of `RSI_ATTESTATION_TOKEN_CONTINUE`, it saves the execution context to the REC object, and performs a REC exit due to IRQ.

During handling of the IRQ, the Host may signal a virtual interrupt to the REC.

On the next entry to the REC, if a virtual interrupt is pending, it is taken to the REC's exception vector.

Whether or not a virtual interrupt was taken, on return to the original thread, the REC determines that `X0` is `RSI_INCOMPLETE`, and therefore calls `RSI_ATTESTATION_TOKEN_CONTINUE` again.



See also:

- [A4.3.5 REC exit due to IRQ](#)
- [A6.1 Realm interrupts](#)
- [A7.2.2 Attestation token generation](#)
- [B5.3.1 RSI_ATTESTATION_TOKEN_CONTINUE command](#)
- [B5.3.2 RSI_ATTESTATION_TOKEN_INIT command](#)
- [D1.3.1 Realm entry and exit flow](#)

D1.8 Realm device assignment flows

See [Chapter A9 Realm device assignment](#).

DRAFT

Chapter D2

Realm shared memory protocol

This section describes a protocol for management of memory which is shared between a Realm and the Host. This protocol makes use of the primitives described in this specification. However, the protocol itself is not part of the RMM architecture. Use of this protocol is subject to a contract between the Realm and Host software agents.

See also:

- [Chapter A5 Realm memory management](#)

D2.1 Realm shared memory protocol description

The Host agrees to provide the Realm with a certain amount of memory. This memory is referred to below as the Realm's "memory footprint".

The memory footprint is described to the Realm, for example via firmware tables. The Realm can choose, at any point during its execution, how much of its memory footprint is protected (accessible only to the Realm) and how much is shared with the Host.

Realm software treats the most significant IPA bit as a "protection attribute" bit. This means that for every Protected IPA (in which the most significant bit is '0'), there exists a corresponding Unprotected IPA alias, which is generated by setting the most significant bit to '1'.

The choice of whether a given page is private or shared at a given time is expressed by setting the RIPAS of the Protected IPA:

- If the RIPAS of the Protected IPA is RAM, the page is private.
- If the RIPAS of the Protected IPA is EMPTY, the page is shared.

The initial RIPAS for every page in the Realm's memory footprint is described to the Realm, for example via firmware tables. The Host agrees that during Realm execution, it will accept a RIPAS change request on any page within the Realm's memory footprint.

Based on the private / shared status of the page, the Host agrees to the following behaviour regarding the Unprotected IPA alias:

- If the page is private, the Host does not create a valid mapping at the Unprotected IPA alias. Realm access to the Unprotected IPA alias causes a REC exit due to Data Abort. In response, the Host sets the "inject_sea" flag on the next REC entry, which causes a Synchronous External Abort to be taken to the Realm.
- If the page is shared, the Host creates (either eagerly, or in response to a REC exit due to Data Abort) a valid mapping at the Unprotected IPA alias. Realm access to the Unprotected IPA alias does not cause a Synchronous External Abort taken to the Realm.

See also:

- [A5.2.1 Realm IPA space](#)
- [A5.2.2 Realm IPA state](#)
- [A5.4 RIPAS change](#)

D2.2 Realm shared memory protocol flow

The following diagram illustrates how the protocol is used to set up and tear down a shared memory buffer.

Chapter D2. Realm shared memory protocol
D2.2. Realm shared memory protocol flow

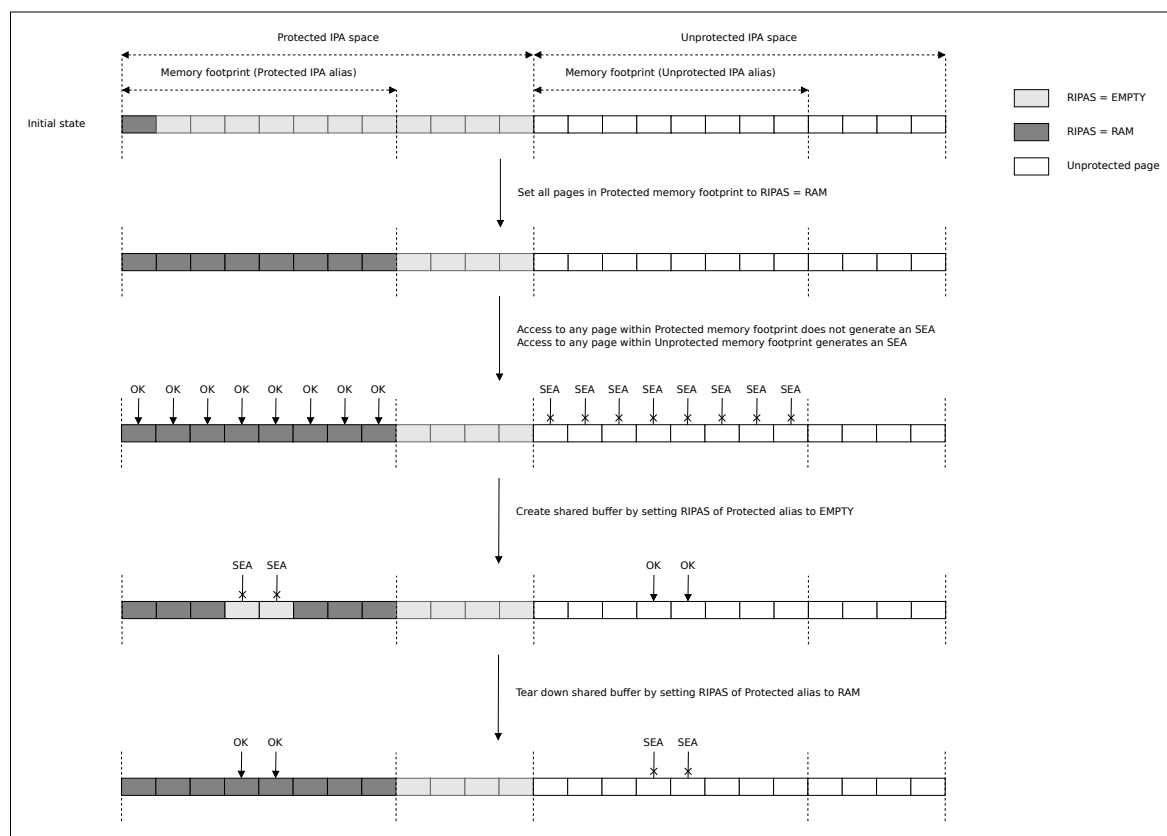


Figure D2.1: Realm shared memory protocol flow

See also:

- [D1.5.3 RIPAS change flow](#)

Glossary

ASL

Arm Specification Language

Language used to express pseudocode implementations. Formal language definition can be found in [Arm Specification Language Reference Manual](#) [18].

CBOR

Concise Binary Object Representation

CCA

Confidential Compute Architecture

CCA platform

All hardware and firmware components which are involved in delivering the CCA security guarantee. See [Arm CCA Security model](#) [4].

CDDL

Concise Data Definition Language

COSE

CBOR Object Signing and Encryption

DOE

Data Object Exchange See [PCI Express 6.0 specification](#) [14]

DSM

Device Security Manager See [PCI Express 6.0 specification](#) [14]

EAT

Entity Attestation Token

ECAM

Enhanced Configuration Access Mechanism See [PCI Express 6.0 specification](#) [14]

FAL

Firmware Activity Log

FID

Function Identifier

GIC

Generic Interrupt Controller

See [Arm Generic Interrupt Controller \(GIC\) Architecture Specification version 3 and version 4](#) [6]

GPF

Granule Protection Fault

GPT

Granule Protection Table
Table which determines the Physical Address Space of each Granule.

HIPAS

Host IPA state

Host

Software executing in Non-secure Security state which manages resources used by Realms

IAK

Initial Attestation Key Key used to sign the CCA platform attestation token.

IDE

Integrity and Data Encryption
See [PCI Express 6.0 specification \[14\]](#)

IPA

Intermediate Physical Address
Address space visible to software executing at EL1 in the Realm.

IPI

Inter-processor interrupt

IRI

Interrupt Routing Infrastructure
A subset of the components which make up the GIC.

ITS

Interrupt Translation Service
A service provided by the GIC.

LFA

Live Firmware Activation

LOR

Limited Order Region
See [Arm Architecture Reference Manual for A-Profile architecture \[3\]](#)

MBZ

Must Be Zero

MEC

Memory Encryption Context

MECID

Memory Encryption Context Identifier

MMIO

Memory-mapped I/O

MPIDR

Multiprocessor Affinity Register

NS

Non-secure

PAS

Physical Address Space

PDEV

Physical Device

Object which represents a communication channel between the RMM and a physical device, for example a PCIe device.

PE

Processing Element

PMU

Performance Monitor Unit

PSCI

Power State Control Interface

See [Arm Power State Coordination Interface \(PSCI\) \[23\]](#)

PSMMU

Physical System Memory Management Unit

See [Arm System Memory Management Unit Architecture Specification \[16\]](#)

RAK

Realm Attestation Key Key used to sign the Realm attestation token.

RD

Realm Descriptor

Object which stores attributes of a Realm.

RDEV

Realm Device

Object which represents Realm view of an assigned device

Realm

A protected execution environment

REC

Realm Execution Context

Object which stores PE state associated with a thread of execution within a Realm.

REM

Realm Extensible Measurement Measurement value which can be extended during the lifetime of a Realm.

RHA

Realm Hash Algorithm

RIM

Realm Initial Measurement Measurement of the state of a Realm at the time of activation.

RIPAS

Realm IPA state

RME

Realm Management Extension

RMI

Realm Management Interface The ABI exposed by the RMM for use by the Host.

RMM

Realm Management Monitor

RNVS

Root Non-volatile Storage

RPV

Realm Personalization Value

RSI

Realm Services Interface The ABI exposed by the RMM for use by the Realm.

RTT

Realm Translation Table
Object which describes the IPA space of a Realm.

RTTE

Realm Translation Table Entry

SBZ

Should Be Zero

SEA

Synchronous External Abort

SGI

Software Generated Interrupt

SMCCC

SMC Calling Convention
See [Arm SMC Calling Convention \[17\]](#)

SMMU

System Memory Management Unit
See [Arm System Memory Management Unit Architecture Specification \[16\]](#)

SPDM

Security Protocol and Data Model
See [Security Protocol and Data Model \(SPDM\) \[19\]](#) and [Secured Messages using SPDM Specification version 1.1.0 \[15\]](#)

SPM

Secure Partition Manager

TA

Trusted Application

TOS

Trusted OS

TSM

Trusted Security Manager

See [Chapter A9 Realm device assignment](#)

VDEV

Virtual Device

Object which represents the binding between a device function and a Realm.

VMM

Virtual Machine Monitor

VMSA

Virtual Memory System Architecture

VPE

Virtual Processing Element

VSMMU

Virtual System Memory Management Unit

See [Arm System Memory Management Unit Architecture Specification](#) [16]

Wiping

An operation which changes the value of a memory location from X to Y , such that the value X cannot be determined from the value Y

DRAFT