Realm Management Monitor specification

Document number	DEN0137
Document quality	BET
Document version	2.0-bet1
Document confidentiality	Non-confidential
Document build information	72673c42 doctool 1.0.0

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.

DRAFT

Quality Status: Beta (BET)

Beta quality status has a particular meaning to Arm of which the recipient must be aware. At this quality level the release is sufficiently stable and committed for initial product development. The recipient can expect some changes to the Beta quality released material.
Please send feedback to the Arm CCA architecture team.

Realm Management Monitor specification

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Release information

2.0-bet1 (30-03-2026)

New features

None

Clarifications

- Correct wording of rule regarding traversal of switches
- Rename RmiPsmmuIrqSet::cmdq to cmdq_sync (FENIMORE-1412)
- VDEV teardown flow: describe failure of refresh / purge operation
- Range-based memory (un)mapping operation targets exclusively Protected or Unprotected IPA space (FENIMORE-1413)
- MEC does not protect memory mapped into Unprotected IPA space (FENIMORE-1416)
- On REC exit due to Emulatable Data Abort, set rec_exit.ISS.SRT to zero (FENIMORE-1356)
- On REC exit due to WFI or WFE, set rec_exit.ISS.RN to zero (FENIMORE-1387)
- Remove reference to SMMU_R_GBPA.ABORT (FENIMORE-1417)
- Move definition of “RMI Address List” into “RMI address range” section (FENIMORE-1449)
- RMI_RTT_{DATA,DEV,UNPROT}_UNMAP: fix description of oaddr input value (FENIMORE-1449)
- Add clarifications related to SROs (FENIMORE-1454)
- RMI_RTT_DATA_MAP: remove redundant statement about wiping (FENIMORE-1414)
- Point out that L1DPT, L1GPT can be allocated with NUMA awareness (FENIMORE-1398)
- RMI_RTT_UNPROT_MAP: correct reference to Unprotected IPA space (FENIMORE-1415)
- Remove unreachable {HIPAS_NARCH_DEV, RIPAS_RAM} from transition diagram (FENIMORE-1455)
- Change to RIPAS of Protected IPA does not modify RIM
- PMU overflow flow: correctly show when overflow interrupt is disabled / enabled
- Remove references to TRACKING_READY, TRACKING_NOT_READY
- Add GRAN_INTERNAL to table of Granule state transitions
- SRO handle: improve consistency of terminology (“invalid” versus “release”)
- RMI_RTT_DEV_VALIDATE: clarify that memory attributes are controlled by the Realm (FENIMORE-1461)
- Remove RMM_STATE_INTERMEDIATE
- RMI_GRANULE_RANGE_{UN}DELEGATE: out_top is valid only when result is RMI_SUCCESS
- Add PSMMU interrupt handling flows (FENIMORE-1470)
- Rename RMI_VSMMU_EVENT_NOTIFY to RMI_VSMMU_EVENT_HANDLE (FENIMORE-1478)
- Document the ignored fields in RmiPdevParams for an RP device (FENIMORE-1465)
- Remove incorrect definition of pdev_id (FENIMORE-1492)
- Specify addr_range in stream connect are CPU system addresses (FENIMORE-1424)
- out_top is valid only when result is RMI_SUCCESS

Defects

- Fix success condition from PDEV_NEW -> PDEV_READY for non-SPDM devices
- Rename negotiation data to protocol data and add a protocol_data_set flag in RsiVdevInfo
- Remove unused RMI_VSMMU_EVENT_COMPLETE (FENIMORE-1421)
- Add RmiRttUnmapFlags::list_count (FENIMORE-1449)
- RMI_GRANULE_RANGE_{UN}DELEGATE: return RMI_ERROR_TRACKING (FENIMORE-1427)
- Remove unused RmiPsmmuIrqResult::empty (FENIMORE-1420)
- Correct implementation of DptEntryCanDescribe() (FENIMORE-1399)
- Add RMI_RMM_STATE_GET (FENIMORE-1388)
- RMI_OP_MEM_{DONATE,RECLAIM}: remove provision for IMPLEMENTATION DEFINED error output values
- RMI_PSMMU_ACTIVATE: return RMI_ERROR_DPT if L0DPT is not created (FENIMORE-1423)
- RMI_VDEV_DESTROY should not be allowed in ERROR state (FENIMORE-1464)
- Reduce the number of addrs returned in PSMMU_IRQ_NOTIFY (FENIMORE-1418)
- Add fatal error return in RMI_PSMMU_IRQ_NOTIFY and change actions to enumeration (FENIMORE-1469)
- Move PDEV stream liveness check from RMI_PDEV_DESTROY to RMI_PDEV_STOP (FENIMORE-1458)
- Allow wired intrs in psmmu activate (FENIMORE-1419)
- RMI_PSMMU_ST_L2_DESTROY: add check on PSMMU state (FENIMORE-1487)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- Specify rid_base, rid_top format is BDF for PCIe devices (FENIMORE-1491)

Relaxations

None

2.0-bet0 (30-01-2026)

New features

- attestation: add Realm Instance ID claim (FENIMORE-1119)

Clarifications

- Consistently use “wipe” instead of “zero-fill” (FENIMORE-919)
- Note that GPF on access to Unprotected IPA is taken to Realm (FENIMORE-920)
- Clarify “RIPAS / HIPAS mismatch” cells in “Summary of Realm IPA space properties” table (FENIMORE-949)
- Clarify that RIPAS / HIPAS are stored in leaf RTT entries (FENIMORE-950)
- Rename “MEC update” to “MEC refresh” (FENIMORE-1211)
- DA: remove misleading arrow regarding verification that device holds private key (FENIMORE-1249)
- P2P: clarify that both coherent and non-coherent device memory is included (FENIMORE-1305)
- RmiRecExitReason: rename RMI_EXIT_VDEV_MAP -> RMI_EXIT_VDEV_VALIDATE_MAPPING (FENIMORE-1336)
- RMI_CMEM_START may cause MEC_COUNT to change (FENIMORE-1376)
- Remove statement that Host must save / restore GIC virtualization system control registers
- TDISP_EN=1 must be set at a given RP when the first PDEV is created (FENIMORE-1358)
- PDEV: abstract device attribute names to be non-PCIe-specific (FENIMORE-1381)
- Add rationale regarding key refresh / purge on VDEV unlock
- RMI_OP_MEM_DONATE: contents of RMI Address List are not modified
- Clarify when the Host should cache the request in the device request buffer
- RIPAS change: include transition to RIPAS_DEV
- For details of extending PAT with CMEM devices, refer to FIRME
- For details of communication with the device via the platform, refer to FIRME
- Clarify Host cached measurements is SPDM measurements transcript excluding VCA
- Clarify value creation of CCA platform software component measurement value in the presence of LFA (FENIMORE-1390)
- Clarify that STE.EATS is set only if PSMMU ats flag is set
- Remove success conditions for RMI_INCOMPLETE (FENIMORE-1400)

Defects

- RMI_PDEV_CREATE: fix initialization of RmmPdev::prot (FENIMORE-1366)
- Tidy up RMI feature registers (FENIMORE-1209)
- RMI_VDEV_COMPLETE: make setting of DEV_COMM_PENDING conditional on intitiating command (FENIMORE-1326)
- RMI_RTT_INIT_RIPAS: don’t extend RIM (FENIMORE-1363)
- RmiAddrRangeDesc: fix position and width of RES0 field (FENIMORE-1378)
- attestation: update profile versions to 2.0.0 (FENIMORE-1377)
- RSI_VDEV_GET_INFO: support report formats other than TDISP (FENIMORE-1329)
- Remove RMI_PDEV_SET_PROT; RMI_VDEV_CREATE fails if PSMMU walk fails (FENIMORE-1375)
- Provide cleaner abstraction between SPDM and non-SPDM devices (FENIMORE-1351)
- Remove measurement indices and always request all device measurements (FENIMORE-1367)
- RMI_PDEV_STREAM_{COMPLETE,KEY_REFRESH,KEY_PURGE}: fix state transitions
- RMI_VDEV_GET_MEASUREMENTS: remove requirement for VDEV to be locked (FENIMORE-1382)
- RMI_REALM_CREATE: bounds check ats_plane parameter (FENIMORE-1385)
- RMI_RTT_INIT_RIPAS: apply change to entries with HIPAS_VOID or HIPAS_DATA (FENIMORE-1350)
- REC exit on access to ICC_*_EL1 is conditional on Host-managed traps (FENIMORE-1338)
- RSI_PLANE_ENTER: rename and reorganise register values (FENIMORE-1210)
- RMI_OP_*: improve consistency of input / output register assignment
- Specify that base of RMI Address List must be aligned to size of an entry
- Specify that when range operation returns out_top, no state change has occurred above that address (FENIMORE-1372)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- Remove requirement for RMM to zero-fill unused output registers
- Add RMI_VSMMU_FEATURES (FENIMORE-1393)
- Add RMI_VSMMU_CMD_{GET,COMPLETE} (FENIMORE-1394)
- RMI_PDEV_CREATE cannot result in caching RMI_DEV_MEASUREMENTS
- RMI_RMM_CONFIG_GET: fail unless RMM state is RMM_ACTIVE (FENIMORE-1401)
- AddrIsTrackingRegionAligned: check against size provided to RMI_RMM_CONFIG_SET
- Remove RmiFeatureRegister0::PGS
- Move RmiFeatureRegister0::{L0GPTSZ,PPS} to RmiFeatureRegister1
- Remove RmmPdev::trust attribute
- Remove unused RmmPdevCoherent type
- Add IDE usage and coherent / I/O coherent traffic to cca-device claim (FENIMORE-1359)
- RMI_PSMMU_DEACTIVATE: fail if L1ST is live (FENIMORE-1404)

Relaxations

- Permit device memory to be either tracked or untracked at platform boot
- Allow RMI_PDEV_ABORT to transition to PDEV_ERROR state

2.0-alp20 (02-01-2026)

New features

None

Clarifications

None

Defects

- RMI_REALM_CREATE: restore failure condition if Realm is live (FENIMORE-1337)
- VDEV measurements: enable signature if supported; always cache SPDM transcript (FENIMORE-1342)
- Add limit on number of VDEVs per PDEV (FENIMORE-1318, FENIMORE-1319)
- Add PA ranges to VDEV (FENIMORE-1320)
- PDEV: separate establishment of communication from establishment of traffic protection (FENIMORE-1343)
- PDEV: add support for NON_TEE stream, directly-attached only (FENIMORE-1328)
- VDEV unlock: add states requiring refresh and purging of keys on all PDEV streams (FENIMORE-1247)
- RMI_RTT_DEV_UNMAP: remove VDEV pointer (FENIMORE-1303)
- Establish binding between Realm attestation token and device attestation evidence (FENIMORE-1023)
- Correct what the Host caches during device measurement retreival (FENIMORE-1352)
- Introduce commands for donating / reclaiming memory used by a stateful RMI operation (FENIMORE-1013)
- Replace “auxiliary Granules” with use of generic donate / reclaim memory pattern (FENIMORE-1297)
- Allow implementation to request additional memory for VSMMU object (FENIMORE-1322)
- Use “donate memory” primitive for allocation of PSMMU stream tables and queues (FENIMORE-1339)
- Use “donate memory” primitive for allocation of granule tracking metadata (FENIMORE-1340)
- Use “donate memory” primitive for allocation of L1GPT (FENIMORE-1340)
- Use “donate memory” primitive for allocation of L0DPT and L1DPT (FENIMORE-1340)
- Permit Host-Side Encryption for CXL type-3 devices (FENIMORE-1278)
- RMI_PDEV_CREATE: defer check for RID range overlap to RMI_PDEV_SET_PROT (FENIMORE-1200)
- Rename HIPAS values (FENIMORE-1134)
  - UNASSIGNED -> VOID
  - ASSIGNED -> DATA
  - ASSIGNED_DEV -> NARCH_DEV
  - ASSIGNED_ARCH_DEV -> ARCH_DEV
  - UNASSIGNED_NS -> UNMAPPED_NS
  - ASSIGNED_NS -> MAPPED_NS

Relaxations

None

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

2.0-alp19 (28-11-2025)

In response to feedback received by Arm on previous 1.1-alp drafts, this release includes some breaking changes to the Realm Management Interface. Consequently, the RMI version is increased from 1.1 to 2.0.

No breaking changes are made to the Realm Services Interface. Consequently, the RSI version remains at 1.1.

This is the first release of the RMM specification in which the RMI and RSI versions do not match. The overall document version number follows the larger of the RMI and RSI versions. Consequently, the document version is increased to 2.0.

Because the 1.1 specification did not reach BET maturity, this draft is effectively a continuation of the previous ALP development sequence. Consequently, this name of this release follow on from the most recent draft (1.1-alp18), and the overall document name is therefore 2.0-alp19.

New features

- Add RMI_GPT_L1_{CREATE,DESTROY} (FENIMORE-1256)
- Replace single-entry memory management commands with range-based equivalents (FENIMORE-1254, FENIMORE-1300)

```
RMI_ARCH_DEV_MAP               -> RMI_RTT_ARCH_DEV_MAP
RMI_ARCH_DEV_UNMAP             -> RMI_RTT_ARCH_DEV_UNMAP
RMI_RTT_AUX_MAP_PROTECTED      -> RMI_RTT_AUX_PROT_MAP
RMI_RTT_AUX_UNMAP_PROTECTED    -> RMI_RTT_AUX_PROT_UNMAP
RMI_RTT_AUX_MAP_UNPROTECTED    -> RMI_RTT_AUX_UNPROT_MAP
RMI_RTT_AUX_UNMAP_UNPROTECTED -> RMI_RTT_AUX_UNPROT_UNMAP
RMI_DATA_CREATE_UNKNOWN        -> RMI_RTT_DATA_MAP
RMI_DATA_DESTROY               -> RMI_RTT_DATA_UNMAP
RMI_VDEV_MAP                   -> RMI_RTT_DEV_MAP
RMI_VDEV_UNMAP                 -> RMI_RTT_DEV_UNMAP
RMI_RTT_MAP_UNPROTECTED        -> RMI_RTT_UNPROT_MAP
RMI_RTT_UNMAP_UNPROTECTED      -> RMI_RTT_UNPROT_UNMAP
```

- More consistent naming of RTT commands (FENIMORE-1304)

```
RMI_DATA_CREATE           -> RMI_RTT_DATA_MAP_INIT
RMI_VDEV_VALIDATE_MAPPING -> RMI_RTT_DEV_VALIDATE
```

- Add “RMM activate” state transition (FENIMORE-1295)

Clarifications

None

Defects

- Bump RMI version to 2.0
- Remove RMI_GRANULE_{UN}DELEGATE (FENIMORE-1300)
- RMI_REALM_CREATE: remove VMID values (FENIMORE-1298)
- Amend statements which still assume RMI Granule size is 4KB (FENIMORE-1075)
- RMI_{PDEV,VDEV}_COMMUNICATE: state constraints on request / response length and offset (FENIMORE-1307)
- Preserve values of GIC CPU interface system registers on REC entry and exit (FENIMORE-1302)
- Add RMI_REALM_TERMINATE (FENIMORE-1152)
- Remove Host control of MECID values and MEC states (FENIMORE-1299)
- Add RsiPlaneExit::elr_el1 (FENIMORE-1308)
- Update cca-platform-extension claim to include IDE, encryption type, VCA (FENIMORE-1327)

Relaxations

None

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

1.1-alp18 (31-10-2025)

New features

- Add RD_AUX granules (FENIMORE-1292)
- VSMMU: add support for stage 1 PRI (FENIMORE-1265)
- Range-based Granule (un)delegation (FENIMORE-1253)

Clarifications

- Replace “resource attested by RMM” with “resource verified by RMM”
- Rename “Initial Attestation Key (IAK)” to “CCA Platform Attestation Key (CPAK)”
- Remove “directly-attached” from the list of CMEM (CXL type-3) device properties
- RMI_GRANULE_{UN}DELEGATE execution time is unbounded due to SMMU RGPTM (FENIMORE-1281)
- Document programming models for RMI and RSI commands
- Describe handling of CPAK X.509 endorsement certificate (FENIMORE-1261)
- Clarify that for CMEM devices, always retrieve signed measurements if supported (FENIMORE-1206)

Defects

- Make “Plane exit due to synchronous exeception” conditional on Plane entry flags
- Initial value of RIM is zero (FENIMORE-1289)
- RSI_VDEV_GET_INFO: return VSMMU address and vSID (FENIMORE-1290)
- RSI_VDEV_DMA_ENABLE: require VDEV state to be VDEV_STARTED (FENIMORE-1291)
- PMU: allow processor Cycle Count to increment during Realm execution (FENIMORE-1287)
- PDEV, VDEV: use similar state machines to reflect communication state (FENIMORE-1283)
- Generalise flow for activating an architectural device such as VSMMU (FENIMORE-1275)
- Remove TRACKING_NONE

Relaxations

None

1.1-alp17 (03-10-2025)

New features

- Add RsiPlaneEnterFlags::trap_dbg (FENIMORE-1268)

Clarifications

- Clarify that Pn has same number of PMU counters, breakpoints, watchpoints as P0 (FENIMORE-1169)
- Add recommendation for calculation of TDISP MMIO Reporting Offset (FENIMORE-1272)

Defects

- Remove RMI_GRANULE_SIZE constant (FENIMORE-1240)
- Remove RmmRtt::entries member (FENIMORE-1240)
- RMI_VDEV_{MAP,UNMAP}: fix shift direction in num_map success conditions (FENIMORE-1241)
- RMI_GRANULE_SIZE_SET: fail if any tracking region has been made tracked (FENIMORE-1243)
- Allow Host to configure tracking region size (FENIMORE-1257)
- State concurrency rules relating to creation / destruction of PSMMU objects (FENIMORE-1167)
- Remove RmiRecExit::rtt_level (FENIMORE-1262)
- Add RmiVdevMeasureFlags::cache_type (FENIMORE-1204)
- Permit granule tracking metadata to be stored in CMEM (FENIMORE-1250)
- Permit DRAM to be untracked at platform boot (FENIMORE-1255)
- Change CWT label for cca-platform-extension-label to be in PSA range (FENIMORE-1277)
- Require platform attestation token to be refreshed following CMEM_START (FENIMORE-1251)
- PSMMU: abstract memory allocation to ST_L1 and queues (FENIMORE-1199)

Relaxations

None

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

1.1-alp16 (04-08-2025)

New features

- Add support for SHA-384 (FENIMORE-1180)
- Add cca-platform-client-id attestation claim (FENIMORE-1166)
- Configurable RMI Granule size (FENIMORE-1075) and runtime allocation of granule metadata (FENIMORE-984)
  - Introduce the concept of a "memory layout view", which represents the RMM's static understanding of PA region reserved for DRAM and device memory.
  - Introduce the concept of a "memory population view", which represents the RMM's dynamic understanding of which PA regions are backed by attested resources.
  - Define "populated memory" to be physical addresses which are backed by attested resources.
  - Introduce the concept of a "memory tracking view", which represents the RMM's dynamic understanding of which PA regions are associated with metadata for recording state information, and the granularity of that tracking.
  - Define "tracked memory" to be physical addresses which have associated tracking metadata.
  - Redefine "delegable memory" as being the combination of "populated" and "tracked".
  - Add the capability to modify the RMI Granule size.
    - Add command RMI_GRANULE_SET_SIZE.
    - This is permitted only while no memory is delegated.
  - Add the capability to allocate Granule tracking metadata at runtime, and therefore transition memory from tracked to untracked:
    - Add command RMI_GRANULE_TRACKING_GET
    - Add command RMI_GRANULE_TRACKING_SET
    - Add command RMI_GRANULE_TRACKING_SIZE
- Add support for coherent memory (CMEM) devices (FENIMORE-986)
  - The only supported type of CMEM device is CXL type-3
  - Add RMI commands used to add CMEM devices to the platform
  - Permit RMM objects and DATA to be allocated from either DRAM or CMEM
  - Modify MEC flows to account for need to propagate key updates to CMEM devices
  - Add "CCA platform extension" claim for reporting attestation information relating to CMEM devices

Clarifications

- Clarify that MAX_MECID is zero if FEAT_MEC is not enabled (FENIMORE-1177)
- For details of IDE key programming, refer to FIRME (FENIMORE-1142)
- Improve description of cca-platform-config attestation claim (FENIMORE-1160)
- Clarify that RmiVdevMeasureParams::nonce is only used for signed measurements (FENIMORE-1021)
- Clarify definition of public key associated with Secure SPDM session (FENIMORE-1175)
- Clarify that Realm measurement is an incremental hash (FENIMORE-1190)
- VDEV graceful teardown flow: set IPA regions to EMPTY (FENIMORE-1158)
- Clarify that transition to Secure SPDM includes verification of device signature (FENIMORE-1176)
- Correct list of Granule states which have an associated owner
- Clarify meaning of "RMI_{PDEV,VDEV}_COMMUNICATE returns COMPLETE" (FENIMORE-1197)
- Clarify meaning of the PSMMU physical address (FENIMORE-1201)

Defects

- Clarify handling of Pn system registers on Plane entry / exit (FENIMORE-1168)
- Enable P0 to inject exception into Pn (FENIMORE-1170)
- Specify that direct interrupt injection is permitted only to the GIC owner Plane (FENIMORE-1171)
- Add RsiPlaneExit::pmu_overflow_status (FENIMORE-1172)
- cca-platform-manufacturing-config attestation claim used incorrect codepoint (FENIMORE-1181)
- cca-realm-mec-policy: use numeric encoding (FENIMORE-1182)
- Remove Realm parameters from RIM (FENIMORE-905)
- RMI_VDEV: consistently pass RD and PDEV to communication commands (FENIMORE-1193)
- RMI_VDEV_GET_MEASUREMENTS: add failure condition for signed measurements not supported (FENIMORE-1021)
- RSI_VDEV_DMA_ENABLE: add non_ats_plane input value (FENIMORE-1178)
- RMI_VDEV_P2P_UNBIND: remove REC argument (FENIMORE-1146)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- RSI_VDEV_GET_INFO: remove p2p_peer (FENIMORE-1148)
- RMI_VDEV_P2P_{BIND,UNBIND}: add failure conditions on VDEV states (FENIMORE-1148)
- Add RSI_VDEV_DMA_DISABLE (FENIMORE-1158)
- Avoid TOCTOU by stating that device configuration is locked at PDEV_READY (FENIMORE-1174)
- RMI_VDEV_GET_MEASUREMENTS: remove RmiVdevMeasureFlags::all (FENIMORE-1183)
- RMI_REC_ENTER: correct check on rec.pending (FENIMORE-1188)
- RMI_{PDEV,VDEV}_COMMUNICATE: allow to return RMI_BUSY for IMPLEMENTATION DEFINED reason (FENIMORE-1191)
- Reinstate Realm verification of device public key digest (FENIMORE-750)

Relaxations

None

1.1-alp15 (11-06-2025)

New features

- Runtime allocation of PSMMU stream tables and queues, and DPT (FENIMORE-987)
- Add new cca-realm-lfa-policy attestation claim (FENIMORE-905)

Clarifications

- Clarify references to MSI by prefixing with physical or virtual (FENIMORE-1138)

Defects

- Make {HIPAS=ASSIGNED_DEV, RIPAS=RAM} unreachable (FENIMORE-1092)
- Replace “independently- / platform-attested” with device properties (FENIMORE-1120)
- Introduce separate feature flags for DA_COH, P2P, VSMMU and ATS (FENIMORE-1049)
- RMI_PDEV_IDE_RESET: require number of VDEVs to be zero (FENIMORE-1050)
- RMI_PDEV_CREATE: restrict IDE SID to [0, 31] (FENIMORE-1044)
- Initial state of MECID zero is SHARED (FENIMORE-1155)
- RSI_VSMMU_ACTIVATE: fix RTTE state check to consider IPA range (FENIMORE-1136)
- RSI_VSMMU_ACTIVATE: prevent reactivation of an active VSMMU (FENIMORE-1135)
- Specify that execute permission for Pn is dropped on transition to RIPAS=DEV (FENIMORE-1157)
- Fix specification of wait / timeout during device communication (FENIMORE-1093)

Relaxations

None

1.1-alp14 (02-05-2025)

New features

- Add cca-platform-manufacturing-config claim to the CCA Platform attestation token (FENIMORE-1037)

Clarifications

- Clarify which Realm sysreg accesses result in a REC exit (FENIMORE-1066)
- RSI_PLANE_SYSREG_{READ,WRITE}: clean up helper functions
- Update IDE key programming flow (FENIMORE-1052)
- RSI_MEASUREMENT_READ: add success conditions (FENIMORE-944)
- PSCI_VERSION: add success condition (FENIMORE-947)
- RSI_IPA_STATE_SET: add success condition (FENIMORE-970)
- RSI_IPA_STATE_GET: add success condition (FENIMORE-957)
- RSI_ATTESTATION_TOKEN_INIT: add success condition (FENIMORE-958)
- RSI_ATTESTATION_TOKEN_CONTINUE: add success condition (FENIMORE-959)
- RIPAS change flow: remove invalid reference to rec.exit_reason (FENIMORE-968)
- Remove inconsistency regarding rec_exit.gprs following emulatable write (FENIMORE-1020)
- Clarify statement regarding successful RMI_REC_ENTER with no forward progress (FENIMORE-1026)
- Add missing diagram arcs for RMI_RTT_INIT_RIPAS (FENIMORE-1048)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- RSI_MEM_SET_PERM_INDEX: add success condition for new_cookie (FENIMORE-1058)
- RSI_VSMMU_ACTIVATE: add success condition (FENIMORE-1059)
- Consistently describe calculation of REC index (FENIMORE-1094)
- PSCI_CPU_ON: add PC to footprint (FENIMORE-1103)
- PSCI_SYSTEM_OFF: add Realm state to footprint (FENIMORE-1104)
- RMI_MEC_SET_PRIVATE: add MEC state to footprint (FENIMORE-1105)
- RMI_VDEV_VALIDATE_MAPPING: add rec.dev_mem_pa to footprint (FENIMORE-1109)
- RSI_MEM_SET_PERM_INDEX: add lock bit to footprint (FENIMORE-1110)
- RSI_MEM_SET_PERM_INDEX: add perm value to footprint (FENIMORE-1111)
- RMI_RTT_SET_S2AP: add rec.s2ap_addr to footprint (FENIMORE-1113)
- RMI_MEC_SET_SHARED: add MEC state to footprint (FENIMORE-1114)
- PSCI_SYSTEM_RESET: add Realm state to footprint (FENIMORE-1116)

Defects

- RealmParamsSupported: check rtt_s2ap_encoding regardless of num_aux_planes (FENIMORE-1053)
- RealmParamsSupported: fix handling of rtt_tree_per_plane and rtt_s2ap_encoding (FENIMORE-1106)
- RMI_PDEV_IDE_RESET: clarify that NCOH IDE connection is targetted (FENIMORE-1061)
- Rename RMI_PDEV_NOTIFY -> RMI_PDEV_STREAM_KEY_REFRESH (FENIMORE-1064)
- RMI_RTT_AUX_MAP_{PROTECTED,UNPROTECTED}: remove fail_index output value (FENIMORE-1101)
- RMI_RTT_AUX_MAP_UNPROTECTED: remove rtt_attr_prot success condition (FENIMORE-1102)
- Share a single RTT among Planes for Unprotected IPA space (FENIMORE-998)
- RMI_RTT_AUX_{MAP,UNMAP}_PROTECTED: remove level(_pri) output value (FENIMORE-1100)
- Alter top level of Attestation token to conform to changes in IETF RATS Conceptual Messages Wrapper (FENIMORE-1065)
- P2P changes (FENIMORE-1036)
  - P2P stream connects exactly two PDEVs

Relaxations

- Relax definition of command observable effects to include memory contents [R_{WZYBV}] (FENIMORE-1112)

1.1-alp13 (18-03-2025)

New features

None

Clarifications

- More clearly state PDEV and VDEV invariants
- Clarify VDEV request flow
- Truncate “IDE setup” flow to remove VDEV interactions
- RMI_DEV_MEM_{MAP,UNMAP} and RMI_RTT_{MAP,UNMAP}_UNPROTECTED: amend language and conditions to accommodate block mappings (FENIMORE-1030)
- {Rmi,Rmm}PdevCoherent: rename IOCOH,FCOH -> NCOH,COH
- Rename RSI_PLANE_REG_{READ,WRITE} -> RSI_PLANE_SYSREG_{READ,WRITE}
- Remove unreachable combinations from HIPAS / RIPAS table
- Show RSI_VSMMU_ACTIVATE on RIPAS state transition diagram
- Clarify meaning of command success pre- and post-conditions
- Prevent VDEV destruction while validated mappings exist (FENIMORE-1017)
- Rename RMI_DEV_MEM_{MAP,UNMAP} -> RMI_VDEV_{MAP,UNMAP} (FENIMORE-1017)
- Rename RMI_RTT_DEV_MEM_VALIDATE -> RMI_VDEV_VALIDATE_MAPPING (FENIMORE-1017)
- For device access to shared memory, use Unprotected IPA as the DMA address (FENIMORE-985)
- Clarify “Physical device setup flow” sequence diagram (FENIMORE-1014)
- Add flows for device measurement retrieval (FENIMORE-1008)
- Add IMPLEMENTATION DEFINED flow for device measurement retrieval (FENIMORE-983)

Defects

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- RMI_PDEV_SET_PUBKEY: correct maximum value for key length (FENIMORE-1031)
- RMI_RTT_DEV_MEM_VALIDATE: add pdev, vdev inputs (FENIMORE-1032)
- RSI_PLANE_SYSREG_{READ,WRITE}: add support for D128
- Add RDEV_OP_STOP
- Remove VDEV instance ID
- Change VDEV communication from Realm-initiated to Host-initiated (FENIMORE-990)
  - Replace the following RSI_RDEV commands with RMI_VDEV equivalents
    - GET_MEASUREMENTS
    - GET_INTERFACE_REPORT
    - LOCK
    - START
  - Remove RSI_RDEV_STOP
  - Add RMI_VDEV_P2P_BIND
  - RmmVdev: add lock_nonce, meas_count, report_count attributes
  - RmmVdev: add op attribute, which records outstanding operation
  - RMI_VDEV_COMMUNICATE: add success conditions which update nonce / counters
  - Remove RSI_RDEV_GET_STATE: the state is now reported via RSI_RDEV_GET_INFO
  - Remove RSI_RDEV_CONTINUE
  - Remove REC exit due to VDEV communication
  - Remove concept of RDEV
  - Remove “Interruptible Realm device operations”
  - Remove RDEV lifecycle diagram
  - Rename remaining RSI_RDEV commands to RSI_VDEV
  - Rewrite overall flow descriptions to reflect Host-initiated requests
  - Add RSI_VDEV_DMA_ENABLE
- RSI_VDEV_GET_INFO: relax alignment requirements (FENIMORE-1033)
- Add RsiPlaneEnterFlags::trap_simd (FENIMORE-1015)
- Add RsiPlane{Enter,Exit}::spsr_el2 (FENIMORE-1015)
- RSI_VSMMU_ACTIVATE: fail if HIPAS != ASSIGNED_VSMMU (FENIMORE-1046)
- Realm entry to Pn with RMI_INJECT_SEA causes Plane exit
- RMI_PDEV_AUX_COUNT: add failure condition for unsupported flags (FENIMORE-1012)
- Change codepoint for cca-realm-mec-policy attestation claim to avoid clash (FENIMORE-1047)
- Introduce VSMMU state machine (FENIMORE-1046)

Relaxations

- Allocate new FIDs in SMCCC for CCA (FENIMORE-1034)

1.1-alp12 (29-01-2025)

New features

None

Clarifications

- Replace RttLevelIsBlockOrPage function with explicit values (FENIMORE-982)
- Add flow which explains how VSMMU mappings are created and validated (FENIMORE-980)
- VSMMU: fix defects in HIPAS / RIPAS diagrams (FENIMORE-991)
- VSMMU: fix defects in liveness description (FENIMORE-994)
- RMI_VSMMU_DESTROY: input VSMMU address is a PA (FENIMORE-995)
- Remove contradiction from statements regarding Unprotected IPA mappings in multi-Plane Realms (FENIMORE-997)
- Merge RmmRdev type into RmmVdev (FENIMORE-1003)
- Clarify that RSI_RDEV_CONTINUE causes REC exit due to VDEV communication (FENIMORE-1009)
- RMI_VDEV_COMMUNICATE: add example of wait flag being returned during SPDM communication (FENIMORE-955)

Defects

- RSI_IPA_STATE_SET: if ripas is not RAM then flags.destroyed is ignored (FENIMORE-988)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- Define Plane exit due to IRQ (FENIMORE-996)
- RMI_REC_ENTER: add “force P0” flag (FENIMORE-989)
- RMI_VSMMU_MAP: add level argument (FENIMORE-993)
- Add missing enum value RmiPdevState::RMI_PDEV_IDE_RESETTING (FENIMORE-999)
- RMI_VSMMU_UNMAP: add PA argument (FENIMORE-1004)
- RMI_RTT_AUX_MAP_PROTECTED: permit mapping of ASSIGNED_VSMMU (FENIMORE-1005)
- RSI_RDEV: return RSI_ERROR_DEVICE in case of incorrect RDEV state (FENIMORE-1010)
- Remove RDEV_STOPPED state (FENIMORE-1011)
- RMI_DEV_MEM_MAP: take account of level when checking address alignment (FENIMORE-1016)
- RMI_DEV_MEM_{MAP,UNMAP}: modify state of contiguous set of Granules (FENIMORE-1016)

Relaxations

- Relax rules regarding SEA on access to Protected IPA, to permit restartable / recoverable errors to be presented to the Realm (FENIMORE-1006)
- Increase maximum number of REC AUX Granules to 32 (FENIMORE-1007)

1.1-alp11 (10-12-2024)

New features

- P2P device communication (FENIMORE-954)

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)
- Specify value of index for RMI_ERROR_{NOT_SUPPORTED,DEVICE} (FENIMORE-973)
- PDEV lifecycle: remove transition from PDEV_NEW to NULL (FENIMORE-974)
- Provide additional examples of “Granule wiping” implementations (FENIMORE-976)
- RSI_RDEV_*: remove orderings between failures with the same error code (FENIMORE-943)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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)
- RsiDevInfo: 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)
- Support for DA with multiple Planes, including ATS (FENIMORE-932)
- Support for signed device measurements (FENIMORE-918)
- RMI_PDEV_DESTROY: remove redundant failure condition pdev_live (FENIMORE-975)
- RMI_RTT_MAP_UNPROTECTED: check bounds of S2AP base index (FENIMORE-977)
- Replace PDEV protection enum with separate spdm, ide and coh flags (FENIMORE-979)
- For platform-attested devices, permit measurements to use an IMPLEMENTATION DEFINED format (FENIMORE-978)

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)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 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)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Relaxations

None

1.0-rel0 (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)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 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 SPDM
  - RMI_PDEV_CREATE: replace device class with flags (SPDM, IDE)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

  - 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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

        - 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)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 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}

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

  - 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

DRAFT

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 [I_{LFYDV}]

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)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

  - 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) [I_{QWQSB}]

Defects

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 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) [I_{GXDDX}]

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 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)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 constistency 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-bet1 (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_{LRCFP}] (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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

– On Non-emulatable Data Abort at an Unprotected IPA, add IL.

Relaxations

None

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 © 2022-2026 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”) . . . xxvii

Preface

Conventions . . . xlvi
Typographical conventions . . . xlvi
Numbers . . . xlvi
Pseudocode descriptions . . . xlvi
Addresses . . . xlvi
Rules-based writing . . . xlvii
Content item identifiers . . . xlvii
Content item rendering . . . xlvii
Content item classes . . . xlvii
Additional reading . . . xlviii
Feedback . . . l
Feedback on this book . . . l
Open issues . . . li

Part A Architecture

Chapter A1 Overview

A1.1 Confidential computing . . . 53
A1.2 System software components . . . 54
A1.3 Realm Management Monitor . . . 54

Chapter A2 Concepts

A2.1 RMM . . . 57
A2.1.1 RMM attributes . . . 57
A2.1.2 RMM lifecycle . . . 57
A2.2 Realm . . . 59
A2.2.1 Overview . . . 59
A2.2.2 Realm execution environment . . . 59
A2.2.3 Realm attributes . . . 60
A2.2.4 Realm liveness . . . 62
A2.2.5 Realm lifecycle . . . 62
A2.2.6 Realm parameters . . . 64
A2.2.7 Realm Descriptor . . . 64
A2.3 Physical memory . . . 65
A2.3.1 Granule size . . . 65
A2.3.2 Views of physical memory . . . 65
A2.3.3 Populated physical memory . . . 66
A2.3.4 Granule tracking region . . . 67
A2.3.5 Delegable physical memory . . . 69
A2.3.6 Granule state . . . 70
A2.3.7 Granule ownership . . . 74
A2.3.8 Granule wiping . . . 75
A2.3.9 Granule Protection Table management . . . 76

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.4 Realm Execution Context . . . 77
A2.4.1 Overview . . . 77
A2.4.2 REC attributes . . . 77
A2.4.3 REC index and MPIDR value . . . 79
A2.4.4 REC lifecycle . . . 79

Chapter A3 Feature discovery and configuration
A3.1 Feature discovery and configuration overview . . . 82
A3.2 Realm hash algorithm . . . 82
A3.3 Realm LPA2 and IPA width . . . 82
A3.4 Realm support for Scalable Vector Extension . . . 83
A3.5 Realm support for self-hosted debug . . . 84
A3.6 Realm support for Performance Monitors Extension . . . 84
A3.7 Realm support for Activity Monitors Extension . . . 84
A3.8 Realm support for Statistical Profiling Extension . . . 84
A3.9 Realm support for Trace Buffer Extension . . . 85
A3.10 Support for Realm device assignment . . . 85
A3.11 Support for auxiliary Planes . . . 85
A3.12 Support for Stage 2 Access Permissions indirect encoding . . . 85
A3.13 Live Firmware Activation . . . 86
A3.14 Support for Realm memory encryption . . . 86

Chapter A4 Realm exception model
A4.1 Realm exception model overview . . . 88
A4.2 REC entry . . . 90
A4.2.1 RmiRecEnter object . . . 90
A4.2.2 General purpose registers restored on REC entry . . . 90
A4.2.3 REC entry following REC exit due to Data Abort . . . 91
A4.3 REC exit . . . 92
A4.3.1 RmiRecExit object . . . 92
A4.3.2 Realm exit reason . . . 93
A4.3.3 General purpose registers saved on REC exit . . . 93
A4.3.4 REC exit due to synchronous exception . . . 94
A4.3.5 REC exit due to IRQ . . . 97
A4.3.6 REC exit due to FIQ . . . 97
A4.3.7 REC exit due to PSCI . . . 97
A4.3.8 REC exit due to RIPAS change pending . . . 98
A4.3.9 REC exit due to Host call . . . 99
A4.3.10 REC exit due to SError . . . 99
A4.3.11 REC exit due to S2AP change pending . . . 99
A4.3.12 REC exit due to VDEV request . . . 100
A4.3.13 REC exit due to VDEV mapping validation . . . 100
A4.3.14 REC exit due to VSMMU command . . . 101
A4.4 Emulated Data Aborts . . . 102
A4.5 Host call . . . 102

Chapter A5 Realm memory management
A5.1 Realm memory management overview . . . 104
A5.2 Realm view of memory management . . . 104
A5.2.1 Realm IPA space . . . 104
A5.2.2 Realm IPA state . . . 104
A5.2.3 Realm access to a Protected IPA . . . 105
A5.2.4 RSI command access to a Protected IPA . . . 105
A5.2.5 Changes to RIPAS while Realm state is REALM_NEW . . . 105
A5.2.6 Changes to RIPAS while Realm state is REALM_ACTIVE . . . 106

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A5.2.7 Realm access to an Unprotected IPA . . . 107
A5.2.8 Synchronous External Aborts . . . 108
A5.2.9 Realm access outside IPA space . . . 108
A5.2.10 Summary of Realm IPA space properties . . . 109
A5.2.11 Cache maintenance operations . . . 109
A5.3 Host view of memory management . . . 110
A5.3.1 Host IPA state . . . 110
A5.3.2 Memory mapping operations overview . . . 111
A5.3.3 Range-based memory operations . . . 112
A5.3.4 Mapping initial Realm image in Protected IPA space . . . 116
A5.3.5 Create mappings from Protected IPA space to wiped conventional memory 116
A5.3.6 Remove mappings from Protected IPA space to conventional memory . 117
A5.3.7 Create mappings from Protected IPA space to device memory . . . 117
A5.3.8 Remove mappings from Protected IPA space to device memory . . . 117
A5.3.9 Create mappings from Protected IPA space to an architectural device . 118
A5.3.10 Remove mappings from Protected IPA space to an architectural device 118
A5.3.11 Create mappings from Unprotected IPA space . . . 118
A5.3.12 Remove mappings from Unprotected IPA space . . . 119
A5.3.13 Create mappings within Protected IPA space in auxiliary RTT tree . . . 119
A5.3.14 Remove mappings within Protected IPA space in auxiliary RTT tree . . 120
A5.3.15 Create mappings within Unprotected IPA space in auxiliary RTT tree . . 120
A5.3.16 Remove mappings within Unprotected IPA space in auxiliary RTT tree . 120
A5.3.17 Changes to HIPAS of a Protected IPA . . . 120
A5.3.18 Summary of changes to HIPAS and RIPAS of a Protected IPA . . . 123
A5.3.19 Dependency of RMI command execution on RIPAS and HIPAS values . 126
A5.3.20 Changes to HIPAS of an Unprotected IPA . . . 127
A5.4 RIPAS change . . . 128
A5.4.1 Realm view of RIPAS change . . . 128
A5.4.2 Host view of RIPAS change to RIPAS_EMPTY or RIPAS_RAM . . . 129
A5.4.3 Host view of RIPAS change to RIPAS_DEV . . . 130
A5.5 VDEV mapping validation . . . 131
A5.6 Realm Translation Table . . . 133
A5.6.1 RTT overview . . . 133
A5.6.2 RTT structure and configuration . . . 133
A5.6.3 RTT starting level . . . 133
A5.6.4 RTT entry . . . 134
A5.6.5 RTT reading . . . 135
A5.6.6 RTT folding . . . 135
A5.6.7 RTT unfolding . . . 136
A5.6.8 RTTE liveness and RTT liveness . . . 136
A5.6.9 RTT destruction . . . 137
A5.6.10 RTT walk . . . 137
A5.6.11 Stage 2 Access Permissions . . . 138
A5.6.12 Memory attributes . . . 140

Chapter A6 Realm interrupts and timers

A6.1 Realm interrupts . . . 145
A6.2 Realm timers . . . 146

Chapter A7 Realm measurement and attestation

A7.1 Realm measurements . . . 149
A7.1.1 Realm Initial Measurement . . . 149
A7.1.2 Realm Extensible Measurement . . . 149
A7.2 Realm attestation . . . 151
A7.2.1 Attestation token . . . 151

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.2 Attestation token generation . . . 151
A7.2.3 Attestation token format . . . 153

Chapter A8 Realm debug and performance monitoring
A8.1 Realm PMU . . . 181

Chapter A9 Realm device assignment
A9.1 Realm device assignment overview . . . 184
A9.1.1 Device objects . . . 184
A9.1.2 Device properties . . . 184
A9.1.3 Device assignment flow . . . 185
A9.2 Physical device object . . . 189
A9.2.1 Physical device attributes . . . 189
A9.2.2 Physical device invariants . . . 191
A9.2.3 Physical device lifecycle . . . 191
A9.2.4 Physical device flows . . . 195
A9.3 Physical device stream object . . . 198
A9.3.1 Physical device stream attributes . . . 198
A9.3.2 Physical device stream invariants . . . 200
A9.3.3 Physical device stream lifecycle . . . 200
A9.4 Virtual device object . . . 206
A9.4.1 Virtual device attributes . . . 206
A9.4.2 Virtual device invariants . . . 207
A9.4.3 Virtual device lifecycle . . . 207
A9.4.4 Mapping from virtual device ID to VDEV object . . . 210
A9.4.5 Virtual device flows . . . 213
A9.5 Communication between RMM and a device . . . 216
A9.5.1 Device requests and responses . . . 216
A9.5.2 Device communication data structures . . . 217
A9.5.3 Host-side device communication flow . . . 220
A9.5.4 Host caching of device attestation evidence . . . 224
A9.5.5 Device measurement retrieval . . . 225
A9.6 Realm management of an assigned virtual device . . . 229
A9.6.1 Realm retrieval of device attestation evidence . . . 229
A9.6.2 Realm validation of device memory mappings . . . 229
A9.6.3 Realm enablement of device DMA . . . 233
A9.7 Physical SMMU . . . 234
A9.7.1 PSMMU object . . . 234
A9.7.2 PSMMU activation . . . 235
A9.7.3 PSMMU deactivation . . . 235
A9.7.4 PSMMU Stream Tables . . . 235
A9.7.5 Device Permission Table . . . 236
A9.7.6 PSMMU interrupts . . . 237
A9.7.7 PSMMU interrupt handling . . . 237
A9.8 Virtual SMMU . . . 240
A9.8.1 VSMMU attributes . . . 240
A9.8.2 VSMMU lifecycle . . . 241
A9.8.3 VSMMU liveness . . . 242
A9.8.4 VSMMU validation . . . 242
A9.8.5 VSMMU commands . . . 244
A9.8.6 Page Request Interface events . . . 245
A9.9 Device memory access . . . 250
A9.9.1 Device access to a Protected IPA . . . 250

Chapter A10 Planes

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A10.1 Planes overview . . . 252
A10.2 Planes exception model . . . 253
A10.2.1 Plane exception model overview . . . 253
A10.2.2 Plane entry . . . 253
A10.2.3 Plane exit . . . 254
A10.2.4 REC exit from Pn . . . 256
A10.2.5 Pn execution of HVC and SMC . . . 256
A10.2.6 Pn system registers . . . 257
A10.2.7 Pn usage of SIMD and SVE . . . 257
A10.3 Planes memory management . . . 258
A10.3.1 Auxiliary RTT . . . 258
A10.3.2 Stage 2 Access Permissions within a multi-Plane Realm . . . 259
A10.4 Planes interrupts . . . 263
A10.5 Planes timers . . . 264
A10.6 Planes debug and performance monitoring . . . 265
A10.6.1 Planes PMU . . . 265
A10.6.2 Planes debug . . . 265

Chapter A11 Realm memory encryption
A11.1 Realm memory encryption overview . . . 267
A11.1.1 MEC and Realms . . . 267

Part B Interface

Chapter B1 Commands
B1.1 Overview . . . 270
B1.2 Command definition . . . 271
B1.2.1 Example command . . . 271
B1.3 Command registers . . . 272
B1.4 Command condition expressions . . . 272
B1.5 Command context values . . . 272
B1.6 Command failure conditions . . . 273
B1.7 Command success conditions . . . 275
B1.8 Concrete and abstract types . . . 275
B1.9 Command footprint . . . 275
B1.10 Command testing . . . 277

Chapter B2 Interface versioning

Chapter B3 Command condition functions
B3.1 AddrInRange function . . . 281
B3.2 AddrIsAligned function . . . 281
B3.3 AddrIsAuxLive function . . . 282
B3.4 AddrIsAuxRef function . . . 282
B3.5 AddrIsProtected function . . . 282
B3.6 AddrIsRmiGranuleAligned function . . . 282
B3.7 AddrIsRsiGranuleAligned function . . . 282
B3.8 AddrIsRttLevelAligned function . . . 283
B3.9 AddrIsTrackingRegionAligned function . . . 283
B3.10 AddrIsWithin function . . . 283
B3.11 AddrRangeIsAuxLive function . . . 283
B3.12 AddrRangeIsProtected function . . . 283
B3.13 AddrRangeIsWithin function . . . 284
B3.14 AddrSetAllDelegableCohDevMem function . . . 284

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.15 AddrSetAllDelegableNonCohDevMem function . . . 284
B3.16 AddrSetEntry function . . . 285
B3.17 AlignDownToRttLevel function . . . 285
B3.18 AlignUpToRttLevel function . . . 285
B3.19 AnyRecRunning function . . . 285
B3.20 AttestationTokenMaxSize function . . . 285
B3.21 AttestationTokenWrite function . . . 286
B3.22 CurrentRealm function . . . 286
B3.23 CurrentRec function . . . 286
B3.24 DeviceCommunicate function . . . 286
B3.25 DptEntryCanDescribe function . . . 286
B3.26 DptL0 function . . . 287
B3.27 DptL0Walk function . . . 287
B3.28 Equal function . . . 287
B3.29 FeatureToRmi function . . . 290
B3.30 FeatureToRsi function . . . 290
B3.31 Gicv3ConfigIsValid function . . . 290
B3.32 GptL0Walk function . . . 291
B3.33 GptL1IsHomogeneous function . . . 291
B3.34 GranuleAt function . . . 291
B3.35 GranulesAllState function . . . 291
B3.36 GranulesAllStateList function . . . 291
B3.37 GranuleSizeFromRmi function . . . 292
B3.38 GranuleSizeToRmi function . . . 292
B3.39 MecidAvailable function . . . 292
B3.40 MemCategoryIsCompatible function . . . 292
B3.41 MemPermLabelSupported function . . . 292
B3.42 MinAddress function . . . 293
B3.43 MpidrEqual function . . . 293
B3.44 MpidrIsUsed function . . . 293
B3.45 MsiAddrIsValid function . . . 293
B3.46 NonSecureAccessPermitted function . . . 293
B3.47 OperationCanCancel function . . . 293
B3.48 OperationIncomplete function . . . 294
B3.49 PaAllDelegableConventional16 function . . . 294
B3.50 PaAllDelegableConventional32 function . . . 294
B3.51 PaIsCohDevMem function . . . 294
B3.52 PaIsDelegable function . . . 295
B3.53 PaIsDelegableCohDevMem function . . . 295
B3.54 PaIsDelegableConventional function . . . 295
B3.55 PaIsDelegableConventionalFine function . . . 295
B3.56 PaIsDelegableDevMem function . . . 296
B3.57 PaIsDelegableNonCohDevMem function . . . 296
B3.58 PaIsDram function . . . 296
B3.59 PaIsNonCohDevMem function . . . 296
B3.60 PaIsPopulated function . . . 297
B3.61 PaIsTracked function . . . 297
B3.62 PaIsTrackedFine function . . . 297
B3.63 PaRangeIsPopulated function . . . 297
B3.64 PaRangeIsUnpopulated function . . . 298
B3.65 PdevAt function . . . 298
B3.66 PdevFlags function . . . 298
B3.67 PdevIsBusy function . . . 298
B3.68 PdevRidRangeOverlap function . . . 298
B3.69 PdevStreamAlloc function . . . 299

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.70 PdevStreamFromHandle function . . . 299
B3.71 PdevStreamFromType function . . . 299
B3.72 PdevStreamLive function . . . 299
B3.73 PdevStreamPdev2Category function . . . 299
B3.74 PdevStreamPdev2Required function . . . 300
B3.75 PdevStreamsForVdev function . . . 300
B3.76 PdevStreamTypeFromRmi function . . . 300
B3.77 PdevVsmmuIsCompatible function . . . 300
B3.78 PlaneSysregValid function . . . 301
B3.79 PlaneSysregValue function . . . 301
B3.80 PsciReturnCodeEncode function . . . 301
B3.81 PsciReturnCodePermitted function . . . 301
B3.82 PsciVersion function . . . 302
B3.83 PsmmuAddrIsValid function . . . 302
B3.84 PsmmuAt function . . . 302
B3.85 PsmmuCmdQueueFull function . . . 302
B3.86 PsmmuFromPdev function . . . 302
B3.87 PsmmuInfoAt function . . . 302
B3.88 PsmmuL1StIsLive function . . . 302
B3.89 PsmmuL2StIsLive function . . . 303
B3.90 PsmmuStWalk function . . . 303
B3.91 PsmmuSupportsMsi function . . . 303
B3.92 RealmAt function . . . 303
B3.93 RealmIpaRangeAllRipasIf function . . . 303
B3.94 RealmIsLive function . . . 303
B3.95 RealmMeasurementEncode function . . . 304
B3.96 RealmParamsSupported function . . . 304
B3.97 RealmRttBaseEqual function . . . 306
B3.98 RecAt function . . . 306
B3.99 RecAuxCount function . . . 306
B3.100 RecDevMemResponseToRsi function . . . 306
B3.101 RecFromMpidr function . . . 307
B3.102 RecIndex function . . . 307
B3.103 RecRipasResponseToRsi function . . . 307
B3.104 RecS2APResponseToRsi function . . . 307
B3.105 RemExtend function . . . 308
B3.106 RimExtendData function . . . 308
B3.107 RimExtendRec function . . . 308
B3.108 RipasToRmi function . . . 308
B3.109 RmiAddrBlockSizeToLevel function . . . 309
B3.110 RmiAddrRangeDescDecode function . . . 309
B3.111 RmiAddrRangesEqual function . . . 310
B3.112 RmiAddrRangesEqual16 function . . . 310
B3.113 RmiAddrRangesEqual8 function . . . 310
B3.114 RmiAddrRangesValid8 function . . . 311
B3.115 RmiDevCommComplete function . . . 311
B3.116 RmiDevCommDataAt function . . . 311
B3.117 RmiFeatureRegister0Decode function . . . 311
B3.118 RmiFeatureRegisterEncode function . . . 312
B3.119 RmiPdevFlagsDecode function . . . 313
B3.120 RmiPdevFlagsSupported function . . . 313
B3.121 RmiPdevParamsAt function . . . 314
B3.122 RmiPdevParamsIsValid function . . . 314
B3.123 RmiPdevStreamParamsAt function . . . 314
B3.124 RmiPdevStreamParamsIsValid function . . . 314

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.125 RmiPsmmuParamsAt function . . . 314
B3.126 RmiPublicKeyParamsAt function . . . 315
B3.127 RmiRealmParamsAt function . . . 315
B3.128 RmiRealmParamsIsValid function . . . 315
B3.129 RmiRecParamsAt function . . . 315
B3.130 RmiRecRunAt function . . . 315
B3.131 RmiRmmConfigAt function . . . 315
B3.132 RmiVdevFlagsDecode function . . . 316
B3.133 RmiVdevMeasureParamsAt function . . . 316
B3.134 RmiVdevParamsAt function . . . 316
B3.135 RmiVdevParamsIsValid function . . . 316
B3.136 RmiVersionHigherIsSupported function . . . 316
B3.137 RmiVersionHighest function . . . 316
B3.138 RmiVersionHighestBelow function . . . 317
B3.139 RmiVersionIsSupported function . . . 317
B3.140 RmiVersionLowerIsSupported function . . . 317
B3.141 RmiVsmmuParamsAt function . . . 317
B3.142 RmiVsmmuParamsIsValid function . . . 317
B3.143 Rmm function . . . 318
B3.144 RmmConfigIsSupported function . . . 318
B3.145 RsiFeatureRegisterEncode function . . . 318
B3.146 RsiHostCallAt function . . . 319
B3.147 RsiPlaneRunAt function . . . 319
B3.148 RsiRealmConfigAt function . . . 319
B3.149 RsiRealmMeasurement function . . . 319
B3.150 RsiVdevInfoAt function . . . 320
B3.151 RsiVersionHigherIsSupported function . . . 320
B3.152 RsiVersionHighest function . . . 320
B3.153 RsiVersionHighestBelow function . . . 320
B3.154 RsiVersionIsSupported function . . . 320
B3.155 RsiVersionLowerIsSupported function . . . 320
B3.156 RttAllEntriesContiguous function . . . 321
B3.157 RttAllEntriesMemAttr function . . . 321
B3.158 RttAllEntriesRipas function . . . 321
B3.159 RttAllEntriesS2AP function . . . 321
B3.160 RttAllEntriesState function . . . 321
B3.161 RttAt function . . . 322
B3.162 RttConfigIsValid function . . . 322
B3.163 RttDescriptorDecode function . . . 322
B3.164 RttDescriptorIsValidForUnprotected function . . . 322
B3.165 RttEntriesInRangeCohDevMem function . . . 322
B3.166 RttEntriesInRangeMemAttr function . . . 323
B3.167 RttEntriesInRangeNonCohDevMem function . . . 323
B3.168 RttEntriesInRangeOutputContiguous function . . . 324
B3.169 RttEntriesInRangeRipas function . . . 324
B3.170 RttEntryAt function . . . 325
B3.171 RttEntryIndex function . . . 325
B3.172 RttEntryStateToRmi function . . . 325
B3.173 RttFold function . . . 325
B3.174 RttIsHomogeneous function . . . 326
B3.175 RttIsLive function . . . 326
B3.176 RttLevelIsStarting function . . . 326
B3.177 RttLevelIsValid function . . . 326
B3.178 RttLevelSize function . . . 326
B3.179 RttMemAttrEqual function . . . 327

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.180 RttS2APEqual function . . . 327
B3.181 RttsAllProtectedEntriesRipas function . . . 327
B3.182 RttsAllProtectedEntriesState function . . . 328
B3.183 RttsAllUnprotectedEntriesState function . . . 328
B3.184 RttsGranuleState function . . . 328
B3.185 RttSkipEntriesIfState function . . . 328
B3.186 RttSkipEntriesUnlessRipas function . . . 328
B3.187 RttSkipEntriesUnlessState function . . . 329
B3.188 RttSkipEntriesUnlessVoidOrData function . . . 329
B3.189 RttSkipEntriesWithRipas function . . . 329
B3.190 RttSkipNonLiveEntries function . . . 330
B3.191 RttsStateEqual function . . . 330
B3.192 RttTreeRangeAllMemAttr function . . . 331
B3.193 RttTreeRangeAllMemAttrEqual function . . . 331
B3.194 RttTreeRangeAllOaddr function . . . 331
B3.195 RttTreeRangeAllOaddrContig function . . . 331
B3.196 RttTreeRangeAllOaddrEqual function . . . 332
B3.197 RttTreeRangeAllOaddrList function . . . 332
B3.198 RttTreeRangeAllS2AP function . . . 332
B3.199 RttTreeRangeAllShareability function . . . 332
B3.200 RttTreeRangeAllShareabilityEqual function . . . 333
B3.201 RttTreeRangeAllState function . . . 333
B3.202 RttWalk function . . . 333
B3.203 RttWalkAnyNotAligned function . . . 333
B3.204 TdiIdIsFree function . . . 334
B3.205 ToAddress function . . . 334
B3.206 ToBits64 function . . . 334
B3.207 TrackingRegionAt function . . . 334
B3.208 TrackingRegionGranularity function . . . 334
B3.209 TrackingRegionIsTracked function . . . 334
B3.210 TrackingRegionSizeFromRmi function . . . 335
B3.211 TrackingRegionSizeToRmi function . . . 335
B3.212 VdevAddrInRange function . . . 336
B3.213 VdevAt function . . . 336
B3.214 VdevAttestInfoEqual function . . . 336
B3.215 VdevFindMapped function . . . 337
B3.216 VdevFromVdevId function . . . 337
B3.217 VdevGenerateNonce function . . . 337
B3.218 VdevIdIsFree function . . . 337
B3.219 VdevSid function . . . 337
B3.220 VdevStateToRsi function . . . 338
B3.221 VersionEqual function . . . 338
B3.222 VmidsAvailable function . . . 338
B3.223 VsidIsFree function . . . 338
B3.224 VsmmuAt function . . . 339
B3.225 VsmmuFeaturesAt function . . . 339
B3.226 VsmmuIsLive function . . . 339

Chapter B4 Realm Management Interface

B4.1 RMI version . . . 341
B4.2 RMI command return codes . . . 342
B4.3 RMI operations . . . 344
B4.3.1 RMI operations overview . . . 344
B4.3.2 Stateful RMI operations . . . 345
B4.3.3 RMI operations which can lead to an intermediate state . . . 354

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.3.4 Object creation and destruction . . . 354
B4.3.5 Range RMI operations . . . 356
B4.4 RMI address range . . . 358
B4.4.1 RMI Address Range Descriptor . . . 358
B4.4.2 RMI Address Range List . . . 358
B4.5 RMI commands . . . 360
B4.5.1 RMI_DPT_L0_CREATE command . . . 363
B4.5.2 RMI_DPT_L0_DESTROY command . . . 365
B4.5.3 RMI_DPT_L1_CREATE command . . . 367
B4.5.4 RMI_DPT_L1_DESTROY command . . . 369
B4.5.5 RMI_FEATURES command . . . 371
B4.5.6 RMI_GPT_L1_CREATE command . . . 372
B4.5.7 RMI_GPT_L1_DESTROY command . . . 374
B4.5.8 RMI_GRANULE_RANGE_DELEGATE command . . . 376
B4.5.9 RMI_GRANULE_RANGE_UNDELEGATE command . . . 378
B4.5.10 RMI_GRANULE_TRACKING_GET command . . . 380
B4.5.11 RMI_GRANULE_TRACKING_SET command . . . 382
B4.5.12 RMI_OP_CANCEL command . . . 384
B4.5.13 RMI_OP_CONTINUE command . . . 385
B4.5.14 RMI_OP_MEM_DONATE command . . . 387
B4.5.15 RMI_OP_MEM_RECLAIM command . . . 389
B4.5.16 RMI_PDEV_ABORT command . . . 391
B4.5.17 RMI_PDEV_COMMUNICATE command . . . 393
B4.5.18 RMI_PDEV_CREATE command . . . 396
B4.5.19 RMI_PDEV_DESTROY command . . . 399
B4.5.20 RMI_PDEV_GET_STATE command . . . 401
B4.5.21 RMI_PDEV_SET_PUBKEY command . . . 403
B4.5.22 RMI_PDEV_STOP command . . . 405
B4.5.23 RMI_PDEV_STREAM_COMPLETE command . . . 407
B4.5.24 RMI_PDEV_STREAM_CONNECT command . . . 410
B4.5.25 RMI_PDEV_STREAM_DISCONNECT command . . . 413
B4.5.26 RMI_PDEV_STREAM_KEY_PURGE command . . . 415
B4.5.27 RMI_PDEV_STREAM_KEY_REFRESH command . . . 418
B4.5.28 RMI_PSCI_COMPLETE command . . . 421
B4.5.29 RMI_PSMMU_ACTIVATE command . . . 425
B4.5.30 RMI_PSMMU_DEACTIVATE command . . . 428
B4.5.31 RMI_PSMMU_EVENT_CONSUME command . . . 430
B4.5.32 RMI_PSMMU_INFO command . . . 431
B4.5.33 RMI_PSMMU_IRQ_NOTIFY command . . . 433
B4.5.34 RMI_PSMMU_ST_L2_CREATE command . . . 435
B4.5.35 RMI_PSMMU_ST_L2_DESTROY command . . . 437
B4.5.36 RMI_REALM_ACTIVATE command . . . 439
B4.5.37 RMI_REALM_CREATE command . . . 441
B4.5.38 RMI_REALM_DESTROY command . . . 445
B4.5.39 RMI_REALM_TERMINATE command . . . 447
B4.5.40 RMI_REC_CREATE command . . . 449
B4.5.41 RMI_REC_DESTROY command . . . 453
B4.5.42 RMI_REC_ENTER command . . . 455
B4.5.43 RMI_RMM_ACTIVATE command . . . 457
B4.5.44 RMI_RMM_CONFIG_GET command . . . 458
B4.5.45 RMI_RMM_CONFIG_SET command . . . 460
B4.5.46 RMI_RMM_STATE_GET command . . . 462
B4.5.47 RMI_RTT_ARCH_DEV_MAP command . . . 463
B4.5.48 RMI_RTT_ARCH_DEV_UNMAP command . . . 466
B4.5.49 RMI_RTT_AUX_CREATE command . . . 469

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.50 RMI_RTT_AUX_DESTROY command . . . 472
B4.5.51 RMI_RTT_AUX_FOLD command . . . 475
B4.5.52 RMI_RTT_AUX_PROT_MAP command . . . 478
B4.5.53 RMI_RTT_AUX_PROT_UNMAP command . . . 481
B4.5.54 RMI_RTT_AUX_UNPROT_MAP command . . . 484
B4.5.55 RMI_RTT_AUX_UNPROT_UNMAP command . . . 487
B4.5.56 RMI_RTT_CREATE command . . . 490
B4.5.57 RMI_RTT_DATA_MAP command . . . 493
B4.5.58 RMI_RTT_DATA_MAP_INIT command . . . 497
B4.5.59 RMI_RTT_DATA_UNMAP command . . . 500
B4.5.60 RMI_RTT_DESTROY command . . . 504
B4.5.61 RMI_RTT_DEV_MAP command . . . 507
B4.5.62 RMI_RTT_DEV_UNMAP command . . . 511
B4.5.63 RMI_RTT_DEV_VALIDATE command . . . 515
B4.5.64 RMI_RTT_FOLD command . . . 519
B4.5.65 RMI_RTT_INIT_RIPAS command . . . 522
B4.5.66 RMI_RTT_READ_ENTRY command . . . 525
B4.5.67 RMI_RTT_SET_RIPAS command . . . 528
B4.5.68 RMI_RTT_SET_S2AP command . . . 531
B4.5.69 RMI_RTT_UNPROT_MAP command . . . 534
B4.5.70 RMI_RTT_UNPROT_UNMAP command . . . 537
B4.5.71 RMI_VDEV_ABORT command . . . 540
B4.5.72 RMI_VDEV_COMMUNICATE command . . . 542
B4.5.73 RMI_VDEV_COMPLETE command . . . 546
B4.5.74 RMI_VDEV_CREATE command . . . 548
B4.5.75 RMI_VDEV_DESTROY command . . . 553
B4.5.76 RMI_VDEV_GET_INTERFACE_REPORT command . . . 556
B4.5.77 RMI_VDEV_GET_MEASUREMENTS command . . . 559
B4.5.78 RMI_VDEV_GET_STATE command . . . 562
B4.5.79 RMI_VDEV_LOCK command . . . 564
B4.5.80 RMI_VDEV_START command . . . 567
B4.5.81 RMI_VDEV_UNLOCK command . . . 570
B4.5.82 RMI_VERSION command . . . 573
B4.5.83 RMI_VSMMU_CMD_COMPLETE command . . . 575
B4.5.84 RMI_VSMMU_CMD_GET command . . . 578
B4.5.85 RMI_VSMMU_CREATE command . . . 580
B4.5.86 RMI_VSMMU_DESTROY command . . . 583
B4.5.87 RMI_VSMMU_EVENT_HANDLE command . . . 585
B4.5.88 RMI_VSMMU_FEATURES command . . . 588
B4.6 RMI types . . . 589
B4.6.1 RmiAddrBlockSize type . . . 589
B4.6.2 RmiAddrRange type . . . 589
B4.6.3 RmiAddrRangeDesc type . . . 589
B4.6.4 RmiAddrRangeDesc16KB type . . . 590
B4.6.5 RmiAddrRangeDesc4KB type . . . 591
B4.6.6 RmiAddrRangeDesc64KB type . . . 592
B4.6.7 RmiAddrSetDesc type . . . 593
B4.6.8 RmiAddrSetList type . . . 594
B4.6.9 RmiBoolean type . . . 594
B4.6.10 RmiContinueBeyond type . . . 595
B4.6.11 RmiContinueFlags type . . . 595
B4.6.12 RmiDataFlags type . . . 596
B4.6.13 RmiDataMeasureContent type . . . 596
B4.6.14 RmiDevCommData type . . . 596
B4.6.15 RmiDevCommEnter type . . . 597

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.16 RmiDevCommExit type . . . 597
B4.6.17 RmiDevCommExitFlags type . . . 598
B4.6.18 RmiDevCommObject type . . . 599
B4.6.19 RmiDevCommProtocol type . . . 600
B4.6.20 RmiDevCommStatus type . . . 600
B4.6.21 RmiEmulatedMmio type . . . 601
B4.6.22 RmiFeature type . . . 601
B4.6.23 RmiFeatureRegister0 type . . . 602
B4.6.24 RmiFeatureRegister1 type . . . 603
B4.6.25 RmiFeatureRegister2 type . . . 604
B4.6.26 RmiFeatureRegister3 type . . . 605
B4.6.27 RmiFeatureRegister4 type . . . 606
B4.6.28 RmiForceP0 type . . . 607
B4.6.29 RmiGranuleSize type . . . 607
B4.6.30 RmiHashAlgorithm type . . . 607
B4.6.31 RmiInjectSea type . . . 608
B4.6.32 RmiInterfaceVersion type . . . 608
B4.6.33 RmiIrqCfg type . . . 609
B4.6.34 RmiLfaPolicy type . . . 609
B4.6.35 RmiMecPolicy type . . . 610
B4.6.36 RmiMemCategory type . . . 610
B4.6.37 RmiOpCanCancel type . . . 610
B4.6.38 RmiOpMemContig type . . . 611
B4.6.39 RmiOpMemDonateReq type . . . 611
B4.6.40 RmiOpMemReq type . . . 612
B4.6.41 RmiOpMemState type . . . 613
B4.6.42 RmiPdevCategory type . . . 613
B4.6.43 RmiPdevFlags type . . . 613
B4.6.44 RmiPdevParams type . . . 614
B4.6.45 RmiPdevSpdm type . . . 615
B4.6.46 RmiPdevState type . . . 615
B4.6.47 RmiPdevStreamFlags type . . . 616
B4.6.48 RmiPdevStreamParams type . . . 616
B4.6.49 RmiPdevStreamType type . . . 617
B4.6.50 RmiPmuOverflowStatus type . . . 617
B4.6.51 RmiPsmmuFlags type . . . 618
B4.6.52 RmiPsmmuInfo type . . . 618
B4.6.53 RmiPsmmuInfoFlags type . . . 619
B4.6.54 RmiPsmmuIrq type . . . 620
B4.6.55 RmiPsmmuIrqEvent type . . . 620
B4.6.56 RmiPsmmuIrqResult type . . . 620
B4.6.57 RmiPsmmuIrqSet type . . . 621
B4.6.58 RmiPsmmuParams type . . . 622
B4.6.59 RmiPublicKeyParams type . . . 622
B4.6.60 RmiRealmFlags0 type . . . 623
B4.6.61 RmiRealmFlags1 type . . . 623
B4.6.62 RmiRealmParams type . . . 624
B4.6.63 RmiRecCreateFlags type . . . 625
B4.6.64 RmiRecEnter type . . . 626
B4.6.65 RmiRecEnterFlags type . . . 627
B4.6.66 RmiRecExit type . . . 627
B4.6.67 RmiRecExitReason type . . . 629
B4.6.68 RmiRecMpidr type . . . 629
B4.6.69 RmiRecParams type . . . 630
B4.6.70 RmiRecRun type . . . 630

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.71 RmiRecRunnable type . . . 631
B4.6.72 RmiResponse type . . . 631
B4.6.73 RmiResult type . . . 631
B4.6.74 RmiResultDataIncomplete type . . . 632
B4.6.75 RmiResultDataLevel type . . . 633
B4.6.76 RmiResultDataNull type . . . 633
B4.6.77 RmiRipas type . . . 634
B4.6.78 RmiRmmConfig type . . . 634
B4.6.79 RmiRmmState type . . . 635
B4.6.80 RmiRttAddrType type . . . 635
B4.6.81 RmiRttAuxBlock type . . . 635
B4.6.82 RmiRttAuxInvalidPri type . . . 636
B4.6.83 RmiRttAuxMapFlags type . . . 636
B4.6.84 RmiRttAuxUnmapFlags type . . . 637
B4.6.85 RmiRttEntryState type . . . 637
B4.6.86 RmiRttPlaneFeature type . . . 638
B4.6.87 RmiRttProtMapFlags type . . . 638
B4.6.88 RmiRttS2APBase type . . . 639
B4.6.89 RmiRttS2APEncoding type . . . 639
B4.6.90 RmiRttUnmapFlags type . . . 640
B4.6.91 RmiRttUnprotMapFlags type . . . 641
B4.6.92 RmiSignatureAlgorithm type . . . 641
B4.6.93 RmiStatusCode type . . . 642
B4.6.94 RmiTrackingRegionState type . . . 643
B4.6.95 RmiTrap type . . . 644
B4.6.96 RmiVdevFlags type . . . 644
B4.6.97 RmiVdevMeasureFlags type . . . 645
B4.6.98 RmiVdevMeasureParams type . . . 645
B4.6.99 RmiVdevMeasureRaw type . . . 646
B4.6.100 RmiVdevParams type . . . 646
B4.6.101 RmiVdevState type . . . 647
B4.6.102 RmiVsmmuCmdFlags type . . . 647
B4.6.103 RmiVsmmuEventFlags type . . . 648
B4.6.104 RmiVsmmuFeatures type . . . 649
B4.6.105 RmiVsmmuFlags type . . . 649
B4.6.106 RmiVsmmuParams type . . . 650

Chapter B5 Realm Services Interface

B5.1 RSI version . . . 652
B5.2 Programming models for RSI operations . . . 652
B5.2.1 Properties of RSI operations . . . 652
B5.2.2 Progress of long-running RSI operations . . . 652
B5.3 RSI command return codes . . . 654
B5.4 RSI commands . . . 655
B5.4.1 RSI_ARCH_DEV_ACTIVATE command . . . 656
B5.4.2 RSI_ATTESTATION_TOKEN_CONTINUE command . . . 658
B5.4.3 RSI_ATTESTATION_TOKEN_INIT command . . . 660
B5.4.4 RSI_FEATURES command . . . 662
B5.4.5 RSI_HOST_CALL command . . . 663
B5.4.6 RSI_IPA_STATE_GET command . . . 665
B5.4.7 RSI_IPA_STATE_SET command . . . 667
B5.4.8 RSI_MEASUREMENT_EXTEND command . . . 669
B5.4.9 RSI_MEASUREMENT_READ command . . . 671
B5.4.10 RSI_MEM_GET_PERM_VALUE command . . . 673
B5.4.11 RSI_MEM_SET_PERM_INDEX command . . . 675

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.12 RSI_MEM_SET_PERM_VALUE command . . . 677
B5.4.13 RSI_PLANE_ENTER command . . . 679
B5.4.14 RSI_PLANE_SYSREG_READ command . . . 681
B5.4.15 RSI_PLANE_SYSREG_WRITE command . . . 683
B5.4.16 RSI_REALM_CONFIG command . . . 685
B5.4.17 RSI_VDEV_DMA_DISABLE command . . . 687
B5.4.18 RSI_VDEV_DMA_ENABLE command . . . 689
B5.4.19 RSI_VDEV_GET_INFO command . . . 691
B5.4.20 RSI_VDEV_VALIDATE_MAPPING command . . . 693
B5.4.21 RSI_VERSION command . . . 696
B5.4.22 RSI_VSMMU_GET_INFO command . . . 698
B5.5 RSI types . . . 700
B5.5.1 RsiArchDevType type . . . 700
B5.5.2 RsiBoolean type . . . 700
B5.5.3 RsiCommandReturnCode type . . . 700
B5.5.4 RsiDevMemCoherent type . . . 701
B5.5.5 RsiDevMemFlags type . . . 701
B5.5.6 RsiDevMemOrdering type . . . 702
B5.5.7 RsiFeature type . . . 702
B5.5.8 RsiFeatureRegister0 type . . . 703
B5.5.9 RsiGicOwner type . . . 703
B5.5.10 RsiHashAlgorithm type . . . 704
B5.5.11 RsiHostCall type . . . 704
B5.5.12 RsiInterfaceVersion type . . . 705
B5.5.13 RsiPlaneEnter type . . . 705
B5.5.14 RsiPlaneEnterFlags type . . . 706
B5.5.15 RsiPlaneExit type . . . 707
B5.5.16 RsiPlaneExitReason type . . . 708
B5.5.17 RsiPlaneRun type . . . 708
B5.5.18 RsiPmuOverflowStatus type . . . 709
B5.5.19 RsiRealmConfig type . . . 709
B5.5.20 RsiResponse type . . . 709
B5.5.21 RsiRipas type . . . 710
B5.5.22 RsiRipasChangeDestroyed type . . . 710
B5.5.23 RsiRipasChangeFlags type . . . 711
B5.5.24 RsiSysregAddress type . . . 711
B5.5.25 RsiTrap type . . . 712
B5.5.26 RsiVdevDmaFlags type . . . 712
B5.5.27 RsiVdevFlags type . . . 713
B5.5.28 RsiVdevInfo type . . . 714
B5.5.29 RsiVdevReportFormatType type . . . 715
B5.5.30 RsiVdevState type . . . 715

Chapter B6 Power State Control Interface

B6.1 PSCI overview . . . 718
B6.2 PSCI version . . . 718
B6.3 PSCI commands . . . 719
B6.3.1 PSCI_AFFINITY_INFO command . . . 720
B6.3.2 PSCI_CPU_OFF command . . . 722
B6.3.3 PSCI_CPU_ON command . . . 723
B6.3.4 PSCI_CPU_SUSPEND command . . . 725
B6.3.5 PSCI_FEATURES command . . . 726
B6.3.6 PSCI_SYSTEM_OFF command . . . 727
B6.3.7 PSCI_SYSTEM_RESET command . . . 728
B6.3.8 PSCI_VERSION command . . . 729

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B6.4 PSCI types . . . 730
B6.4.1 PsciInterfaceVersion type . . . 730
B6.4.2 PsciReturnCode type . . . 730

Part C Constants and types

Chapter C1 RMM constants

C1.1 RMM_NUM_PERM_OVERLAY_INDICES . . . 733
C1.2 RMM_RTT_PAGE_LEVEL . . . 733
C1.3 RMM_RTT_TREE_PRIMARY . . . 733
C1.4 RSI_GRANULE_SIZE . . . 733

Chapter C2 RMM types

C2.1 RmmAddrRange type . . . 734
C2.2 RmmBoolean type . . . 734
C2.3 RmmDataFlags type . . . 735
C2.4 RmmDataMeasureContent type . . . 736
C2.5 RmmDevCommState type . . . 736
C2.6 RmmDevMemCoherent type . . . 736
C2.7 RmmDevMemFlags type . . . 737
C2.8 RmmDevMemOrdering type . . . 737
C2.9 RmmDptL0 type . . . 737
C2.10 RmmDptL0Entry type . . . 738
C2.11 RmmDptL0EntryState type . . . 738
C2.12 RmmDptL0State type . . . 738
C2.13 RmmFeature type . . . 739
C2.14 RmmGlobal type . . . 739
C2.15 RmmGlobalDynamic type . . . 739
C2.16 RmmGlobalStatic type . . . 740
C2.17 RmmGptL0Entry type . . . 741
C2.18 RmmGptL0EntryState type . . . 741
C2.19 RmmGranule type . . . 742
C2.20 RmmGranuleState type . . . 742
C2.21 RmmHashAlgorithm type . . . 743
C2.22 RmmHipas type . . . 743
C2.23 RmmIrqCfg type . . . 743
C2.24 RmmLfaPolicy type . . . 744
C2.25 RmmMeasurementDescriptorData type . . . 744
C2.26 RmmMeasurementDescriptorRec type . . . 745
C2.27 RmmMecPolicy type . . . 745
C2.28 RmmMemCategory type . . . 746
C2.29 RmmMemPermLocked type . . . 746
C2.30 RmmMemPerms type . . . 746
C2.31 RmmOpCanCancel type . . . 747
C2.32 RmmPdev type . . . 747
C2.33 RmmPdevCategory type . . . 748
C2.34 RmmPdevOperation type . . . 748
C2.35 RmmPdevSpdm type . . . 748
C2.36 RmmPdevState type . . . 749
C2.37 RmmPdevStream type . . . 749
C2.38 RmmPdevStreamResult type . . . 750
C2.39 RmmPdevStreamState type . . . 750
C2.40 RmmPdevStreamType type . . . 751
C2.41 RmmPsmmu type . . . 751

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

C2.42 RmmPsmmuState type . . . 752
C2.43 RmmPsmmuStEntry type . . . 752
C2.44 RmmPsmmuStEntryState type . . . 753
C2.45 RmmPsmmuStWalkResult type . . . 753
C2.46 RmmReadWriteOp type . . . 753
C2.47 RmmRealm type . . . 753
C2.48 RmmRealmMeasurement type . . . 755
C2.49 RmmRealmState type . . . 755
C2.50 RmmRec type . . . 755
C2.51 RmmRecAttestState type . . . 756
C2.52 RmmRecEmulatableAbort type . . . 757
C2.53 RmmRecFlags type . . . 757
C2.54 RmmRecPending type . . . 757
C2.55 RmmRecResponse type . . . 758
C2.56 RmmRecRunnable type . . . 758
C2.57 RmmRecState type . . . 759
C2.58 RmmRipas type . . . 759
C2.59 RmmRipasChangeDestroyed type . . . 759
C2.60 RmmRtt type . . . 760
C2.61 RmmRttEntry type . . . 760
C2.62 RmmRttEntryState type . . . 761
C2.63 RmmRttMemAttr type . . . 762
C2.64 RmmRttPlaneFeature type . . . 762
C2.65 RmmRttProtected type . . . 762
C2.66 RmmRttS2APBase type . . . 763
C2.67 RmmRttS2APDirect type . . . 763
C2.68 RmmRttS2APEncoding type . . . 763
C2.69 RmmRttS2APIndirect type . . . 764
C2.70 RmmRttShareability type . . . 764
C2.71 RmmRttWalkNotAligned type . . . 764
C2.72 RmmRttWalkResult type . . . 765
C2.73 RmmSmmuMsiConfig type . . . 765
C2.74 RmmState type . . . 766
C2.75 RmmSystemRegisters type . . . 766
C2.76 RmmTrackingRegion type . . . 766
C2.77 RmmTrackingRegionState type . . . 767
C2.78 RmmVdev type . . . 767
C2.79 RmmVdevAddrResult type . . . 768
C2.80 RmmVdevAttestInfo type . . . 768
C2.81 RmmVdevDmaState type . . . 768
C2.82 RmmVdevOperation type . . . 769
C2.83 RmmVdevState type . . . 769
C2.84 RmmVsmmu type . . . 770
C2.85 RmmVsmmuState type . . . 770

Chapter C3 Generic types

C3.1 Address type . . . 771
C3.2 BitsN type . . . 771
C3.3 IntN type . . . 771
C3.4 UIntN type . . . 772

Part D Usage

Chapter D1 Flows

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

D1.1 Granule delegation flows . . . 775
D1.1.1 Granule delegation flow . . . 775
D1.1.2 Granule undelegation flow . . . 775
D1.2 Realm lifecycle flows . . . 777
D1.2.1 Realm creation flow . . . 777
D1.2.2 Realm Translation Table creation flow . . . 777
D1.2.3 Initialize memory of New Realm flow . . . 778
D1.2.4 REC creation flow . . . 780
D1.2.5 Realm destruction flow . . . 781
D1.3 Realm exception model flows . . . 784
D1.3.1 Realm entry and exit flow . . . 784
D1.3.2 Host call flow . . . 784
D1.3.3 REC exit due to Data Abort fault flow . . . 785
D1.3.4 MMIO emulation flow . . . 786
D1.4 PSCI flows . . . 788
D1.4.1 PSCI_CPU_ON flow . . . 788
D1.5 Realm memory management flows . . . 791
D1.5.1 Add memory to Active Realm flow . . . 791
D1.5.2 NS memory flow . . . 791
D1.5.3 RIPAS change flow . . . 792
D1.5.4 S2AP change flow . . . 793
D1.6 Realm interrupts and timers flows . . . 794
D1.6.1 Interrupt flow . . . 794
D1.6.2 Timer interrupt delivery flow . . . 794
D1.7 Realm attestation flows . . . 796
D1.7.1 Attestation token generation flow . . . 796
D1.7.2 Handling interrupts during attestation token generation flow . . . 797
D1.8 Realm device assignment flows . . . 799

Chapter D2 Realm shared memory protocol
D2.1 Realm shared memory protocol description . . . 801
D2.2 Realm shared memory protocol flow . . . 801

Glossary

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Preface

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

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

DRAFT

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_{iiiii}

- *L* is a label that indicates the content class of the content item.
- *iiiii* 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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*.

DRAFT

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] *RATS Conceptual Messages Wrapper (CMW)*. See https://datatracker.ietf.org/doc/draft-ietf-rats-msg-wrap/

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

[8] *CBOR Object Signing and Encryption (COSE) Header Parameters for Carrying and Referencing X.509 Certificates*. See https://datatracker.ietf.org/doc/html/rfc9360.html

[9] *Concise Binary Object Representation (CBOR)*. See https://tools.ietf.org/html/rfc7049

[10] *CBOR Object Signing and Encryption (COSE)*. See https://tools.ietf.org/html/rfc8152

[11] *Entity Attestation Token (EAT)*. See https://datatracker.ietf.org/doc/draft-ietf-rats-eat/

[12] *Concise Data Definition Language (CDDL)*. See https://tools.ietf.org/html/rfc8610

[13] *IANA Named Information Hash Algorithm Registry*. See http://www.iana.org/assignments/named-information

[14] *SEC 1: Elliptic Curve Cryptography, version 2.0*. See https://www.secg.org/sec1-v2.pdf

[15] *RME system architecture spec*. (ARM DEN 0129) Arm Ltd.

[16] *PCI Express 6.0 specification*. See https://pcisig.com/pci-express-6.0-specification

[17] *Secured Messages using SPDM Specification version 1.1.0*. See https://www.dmtf.org/sites/default/files/standards/documents/DSP0277_1.1.0.pdf

[18] *Compute eXpress Link specification*. See https://computeexpresslink.org/cxl-specification/

[19] *Firmware Interfaces for RME (FIRME) specification*. (ARM DEN 0149) Arm Limited.

[20] *Realm Host Interface specification*. (ARM DEN 0148) Arm Limited.

[21] *Security Protocol and Data Model (SPDM)*. See https://www.dmtf.org/dsp/DSP0274

[22] *Arm System Memory Management Unit Architecture Specification*. (ARM IHI 0070) Arm Limited.

[23] *Arm SMC Calling Convention*. (ARM DEN 0028 D) Arm Ltd.

[24] *Arm Specification Language Reference Manual*. (ARM DDI 0612 00bet7) Arm Ltd.

[25] *Secure Hash Standard (SHS)*. See https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[26] *RSA Cryptography Specifications Version 2.2*. See https://datatracker.ietf.org/doc/rfc8017/

[27] *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*. See https://datatracker.ietf.org/doc/html/rfc6979

[28] *Arm Power State Coordination Interface (PSCI)*. (ARM DEN 0022 D.b) Arm Ltd.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 2.0-bet1).
- 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Open issues

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

Key	Description
FENIMORE-1361	Text is missing from bitfield diagrams.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

DRAFT

Part A
Architecture

Chapter A1
Overview

DRAFT

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:

- A2.2 *Realm*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A1.2 System software components

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

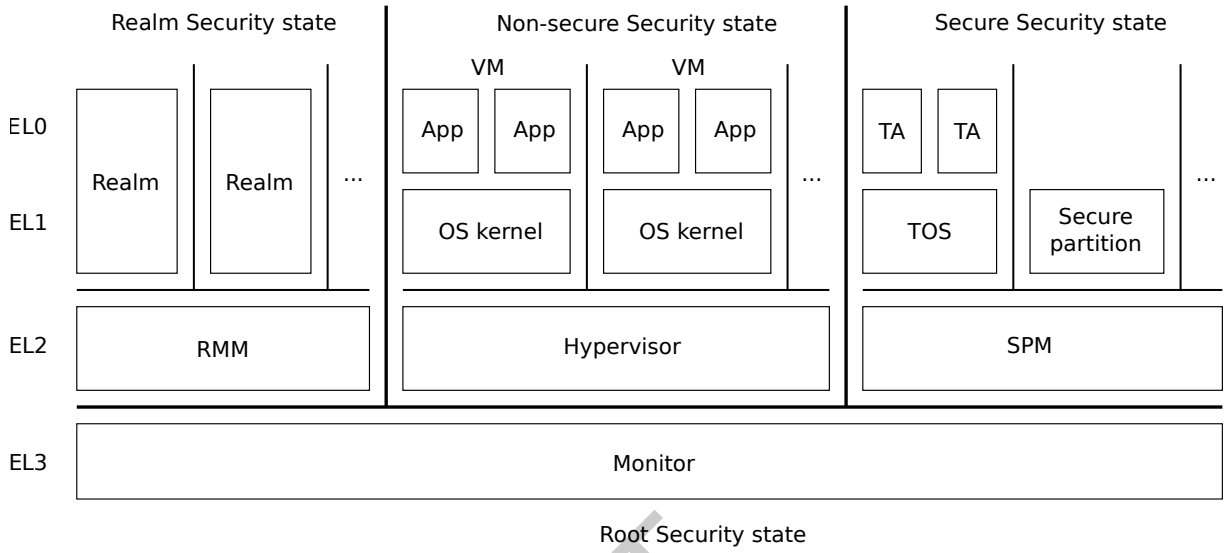


Figure A1.1: System software architecture

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:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 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*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A2
Concepts

DRAFT

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

- A2.1 *RMM*
- A2.2 *Realm*
- A2.3 *Physical memory*
- A2.4 *Realm Execution Context*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.1 RMM

This section describes the global attributes and the lifecycle of the RMM.

A2.1.1 RMM attributes

I_{RHWMT} The global state of the RMM is modelled as comprising two components:

- *Static attributes*
  - Which optional features are supported by the implementation
  - Configuration values of the implementation
- *Dynamic attributes*
  - State which can be mutated through execution of RMM commands

I_{WJRJZ} The Rmm() function returns an RmmGlobal data structure which models both the static and dynamic attributes of the implementation.

D_{LPNGM} The attributes of the RMM are summarized in the following table.

Name	Type	Description
static	RmmGlobalStatic	Static attributes
dynamic	RmmGlobalDynamic	Dynamic attributes

I_{MZQZX} Static attributes of the RMM can be read by execution of RMI_FEATURES.

I_{QMGZX} Dynamic attributes of the RMM can be read by execution of RMI_RMM_CONFIG_GET.

I_{LLTDW} Dynamic attributes of the RMM can be modified by execution of RMI_RMM_CONFIG_SET.

See also:

- A2.3.1 *Granule size*
- A2.3.4 *Granule tracking region*
- Chapter A3 *Feature discovery and configuration*
- B3.143 *Rmm function*
- B4.5.5 *RMI_FEATURES command*
- B4.5.44 *RMI_RMM_CONFIG_GET command*
- B4.5.45 *RMI_RMM_CONFIG_SET command*
- C2.14 *RmmGlobal type*

A2.1.2 RMM lifecycle

A2.1.2.1 States

D_{XFMQC} The states of a RMM are listed below.

State	Description
RMM_STATE_INIT	Initial state of the RMM.
RMM_STATE_ACTIVE	RMM is active.

I_{FDHLK} RMI_RMM_CONFIG_SET fails with RMI_ERROR_GLOBAL unless the RMM state is RMM_STATE_INIT.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{TNTTB} RMI_GRANULE_TRACKING_SET fails with RMI_ERROR_GLOBAL unless the RMM state is RMM_STATE_ACTIVE.

I_{HMVNJ} RMI_GRANULE_TRACKING_GET fails with RMI_ERROR_GLOBAL unless the RMM state is RMM_STATE_ACTIVE.

I_{SFCDR} RMI_GRANULE_RANGE_DELEGATE fails with RMI_ERROR_GLOBAL unless the RMM state is RMM_STATE_ACTIVE. This means that any RMM operation which requires access to delegated memory is also prevented unless the RMM state is RMM_STATE_ACTIVE.

See also:

- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*
- B4.5.11 *RMI_GRANULE_TRACKING_SET command*
- B4.5.45 *RMI_RMM_CONFIG_SET command*

A2.1.2.2 State transitions

I_{HPMDX} Permitted RMM 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
RMM_STATE_INIT	RMM_STATE_ACTIVE	RMI_RMM_ACTIVATE

I_{HWCTN} Permitted RMM state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

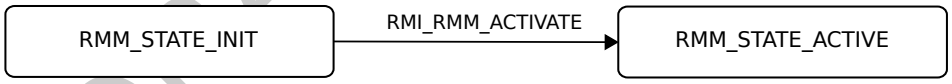


Figure A2.1: RMM state transitions

DRAFT

See also:

- B4.5.43 *RMI_RMM_ACTIVATE command*

See also:

- B4.5.46 *RMI_RMM_STATE_GET command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.2 Realm

This section describes the concept of a Realm.

A2.2.1 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.

A2.2.2 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].

A2.2.2.1 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:

- A2.4 *Realm Execution Context*
- A4.5 *Host call*
- B4.5.37 *RMI_REALM_CREATE command*

A2.2.2.2 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*

A2.2.2.3 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*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.2.2.4 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.

A2.2.3 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
rim	RmmRealmMeasurement	Realm Initial Measurement
rem	RmmRealmMeasurement[4]	Realm Extensible Measurement
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
rpv	Bits512	Realm Personalization Value
feat_da	RmmFeature	Whether Realm device assignment is enabled for this Realm
feat_ats	RmmFeature	Whether Address Translation Service is supported for devices assigned to the Realm
ats_plane	UInt64	Index of Plane whose stage 2 permissions are observed by ATS requests from devices assigned to the 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
mec_policy	RmmMecPolicy	MEC policy

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
num_recs	UInt64	Number of RECs owned by this Realm
num_vdevs	UInt64	Number of VDEVs owned by this Realm
num_vsmmus	UInt64	Number of VSMMUs owned by 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.

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.

I_{GXKDQ} 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_VDEV command fails.

See also:

- A2.2.5 *Realm lifecycle*
- A2.4 *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*
- A7.2.3.1.4 *Realm Personalization Value claim*
- B4.5.74 *RMI_VDEV_CREATE command*
- B5.4.4 *RSI_FEATURES command*
- B5.4.16 *RSI_REALM_CONFIG command*
- C2.47 *RmmRealm type*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.2.4 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_{RLSNX} In order to destroy a Realm, it must first be terminated. Terminating a Realm causes it to enter REALM_ZOMBIE state.

X_{NJWWV} Being in REALM_ZOMBIE state guarantees that a Realm's activity has been quiesced.

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_{YXBPC} A Realm can be both live and in REALM_ZOMBIE state.

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.

I_{PVPQB} If a Realm is live, it cannot be destroyed.

See also:

- A2.2.5 *Realm lifecycle*
- A2.3.6 *Granule state*
- A2.3.7 *Granule ownership*
- A2.4 *Realm Execution Context*
- A5.6.8 *RTTE liveness and RTT liveness*
- A9.4 *Virtual device object*
- A9.8 *Virtual SMMU*
- B3.94 *RealmIsLive function*
- B4.5.38 *RMI_REALM_DESTROY command*
- B4.5.39 *RMI_REALM_TERMINATE command*

DRAFT

A2.2.5 Realm lifecycle

See also:

- Chapter A3 *Feature discovery and configuration*
- D1.2 *Realm lifecycle flows*

A2.2.5.1 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.
REALM_ZOMBIE	Ready for destruction. Not eligible for execution.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.2.5.2 State transitions

I_{RRHFG} 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
NULL	REALM_NEW	RMI_REALM_CREATE
REALM_NEW	REALM_ACTIVE	RMI_REALM_ACTIVATE
REALM_ACTIVE	REALM_SYSTEM_OFF	PSCI_SYSTEM_OFF PSCI_SYSTEM_RESET
REALM_NEW	REALM_ZOMBIE	RMI_REALM_TERMINATE
REALM_ACTIVE	REALM_ZOMBIE	RMI_REALM_TERMINATE
REALM_SYSTEM_OFF	REALM_ZOMBIE	RMI_REALM_TERMINATE
REALM_ZOMBIE	*NULL*	RMI_REALM_DESTROY

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.

DRAFT

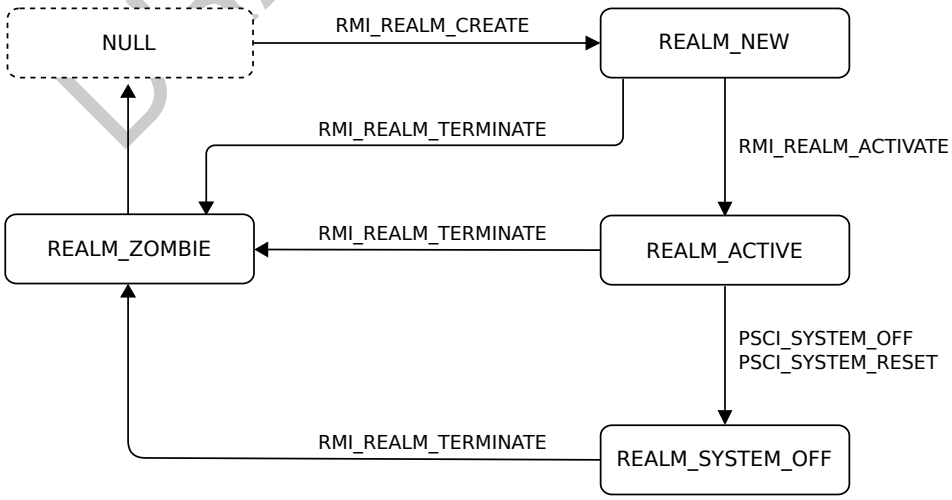


Figure A2.2: Realm state transitions

See also:

- B6.3.6 *PSCI_SYSTEM_OFF command*
- B6.3.7 *PSCI_SYSTEM_RESET command*
- B4.5.36 *RMI_REALM_ACTIVATE command*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.5.38 *RMI_REALM_DESTROY command*
- B4.5.39 *RMI_REALM_TERMINATE command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.2.6 Realm parameters

D_{TGMVZ} A *Realm parameter* is a value which is provided by the Host during Realm creation.

See also:

- A2.2.3 *Realm attributes*
- Chapter A3 *Feature discovery and configuration*
- B3.127 *RmiRealmParamsAt function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.62 *RmiRealmParams type*

A2.2.7 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:

- A2.2.3 *Realm attributes*
- A2.3.6 *Granule state*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.3 Physical memory

This section describes how the RMM manages the usage within Realm PAS of physically memory-mapped resources.

I_{DJGZW} Physically memory-mapped resources may be used within Realm PAS for the following purposes:

- To store code or data used by a Realm
- As device memory used by a Realm
- To store data used by the RMM to manage a Realm

A2.3.1 Granule size

This section describes the granularities at which physical memory may be addressed by the RMM, and via RMM interfaces.

D_{RFGXP} *Physical Granule size* is the smallest unit of physical memory which can be described in a Granule Protection Table (GPT) entry.

D_{PQBNY} *RMI Granule size* is the smallest unit of physical memory for which the RMM manages usage within Realm PAS.

R_{FVRPQ} At platform boot, RMI Granule size is equal to Physical Granule size.

I_{YHZZY} The current RMI Granule size can be discovered by execution of RMI_RMM_CONFIG_GET.

I_{RDXRC} The set of supported RMI Granule sizes can be discovered by execution of RMI_FEATURES.

R_{CSXGJ} The set of supported RMI Granule sizes does not include any value which is not a supported translation granule size.

R_{RVZCL} The set of supported RMI Granule sizes does not include any value which is smaller than the Physical Granule size.

I_{LXSCG} RMI Granule size can be modified by execution of RMI_RMM_CONFIG_SET.

I_{HJRFP} Modification of RMI Granule size fails if any tracking region has been transitioned from untracked to tracked.

D_{VDXKJ} *RSI Granule size* is the smallest unit of physical memory which can be referred to by the input values of an RSI command.

R_{PJWPJ} RSI Granule size is equal to 4KB.

See also:

- A2.3.2 *Views of physical memory*
- A2.3.4 *Granule tracking region*
- B4.5.5 *RMI_FEATURES command*
- B4.5.44 *RMI_RMM_CONFIG_GET command*
- B4.5.45 *RMI_RMM_CONFIG_SET command*

A2.3.2 Views of physical memory

I_{SZPKM} The RMM manages the usage within Realm PAS of physically memory-mapped resources by combining the following views:

- The *memory layout view*, which is a static description of how regions of physical address space are reserved for specific purposes.
- The *memory population view*, which is a dynamic record of the regions of physical address space which are backed by resources which have been verified by the RMM.
- The *memory tracking view*, which is a dynamic record of the regions of physical address space for which the RMM is tracking usage within Realm PAS.

I_{NJQRV} The *memory layout view* consists of information about the system memory map which is known to the RMM at platform boot:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- The physical address size
- Configuration of the Granule Protection Table (GPT)
  - The Level 0 GPT Size (L0GPTSZ)
  - The Physical Granule Size (PGS)
- The physical address region(s) which are reserved for DRAM
- The physical address region(s) which are reserved for coherent device memory
- The physical address region(s) which are reserved for non-coherent device memory

I_{SLDFL} The following diagram shows an example of the system memory map information held by the RMM.

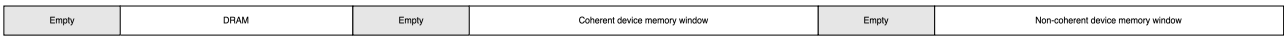


Figure A2.3: Example memory layout view

I_{JGYML} At runtime, the Host can inform the RMM of the existence of a device which is to be made accessible via Realm PAS.

For each such device, the *memory population view* records the following information:

- The physical address regions which are associated with the device. The RMM checks that these lie within the appropriate regions of the system memory map, and that they do not overlap with those associated with any other device.
- Whether the RMM has verified the identity and configuration of the device.

I_{RPDVX} In order to track the usage within Realm PAS of physically memory-mapped resources, the RMM requires memory for storage of Granule metadata. This metadata includes Granule state, and may also include additional IMPLEMENTATION DEFINED information.

To enable tracking for a given region of physical address space, the Host specifies the granularity at which Granule state should be tracked for that region, and provides sufficient memory to the RMM for storage of the metadata.

For each region of physical address space, the *memory tracking view* records whether tracking has been enabled, and if so at which granularity.

DRAFT

See also:

- *Arm Architecture Reference Manual for A-Profile architecture* [3]
- A2.3.3 *Populated physical memory*
- A2.3.4 *Granule tracking region*
- Chapter A9 *Realm device assignment*
- B3.51 *PaIsCohDevMem function*
- B3.58 *PaIsDram function*
- B3.59 *PaIsNonCohDevMem function*
- B3.60 *PaIsPopulated function*

A2.3.3 Populated physical memory

D_{HGSSR} A physical address is *populated* if it is backed by a resource which has been verified by the RMM.

R_{KGHRC} An address which is within a region of the system memory map that is reserved for DRAM is *populated*. DRAM hot-plug is not supported.

R_{FFWHK} Device memory is *populated* if the state of the corresponding PDEV is PDEV_READY.

I_{NSJLW} The following diagram shows an example of how the memory population view is updated following attestation of the identity and configuration of devices.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

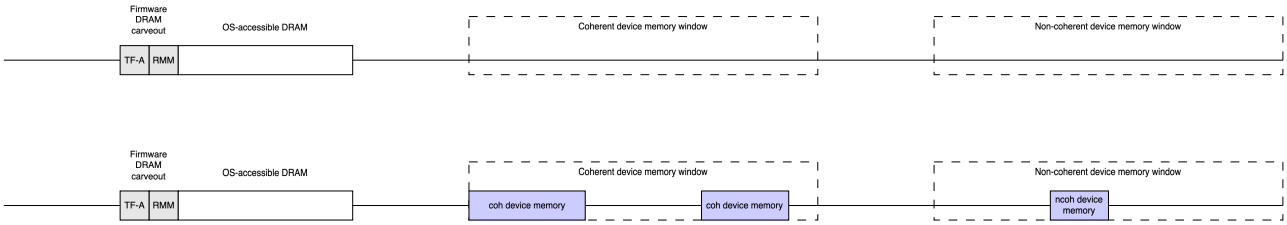


Figure A2.4: Example memory population view

See also:

- A2.3.2 *Views of physical memory*
- Chapter A9 *Realm device assignment*

A2.3.4 Granule tracking region

D_{DFYKK} A *Granule tracking region* is a naturally-aligned region of physical address space.

I_{MGKPN} The Granule tracking region size can be discovered by execution of RMI_RMM_CONFIG_GET.

I_{VBKCK} The Granule tracking region size can be modified by execution of RMI_RMM_CONFIG_SET.

R_{FQBJD} The valid combinations of RMI Granule size and Granule tracking region size are listed in the following table.

RMI Granule size	Granule tracking region sizes
4KB	1GB
16KB	32MB, 64GB
64KB	512MB, 4TB

DRAFT

D_{WBKYM} The attributes of a Granule tracking region are summarized in the following table.

Name	Type	Description
state	RmmTrackingRegionState	Tracking region state.
category	RmmMemCategory	Memory category.

D_{PRNVF} The states of a Granule tracking region are listed below.

Name	Description
TRACKING_COARSE	Region is tracked at the granularity of the Tracking Region size.
TRACKING_FINE	Region is tracked at the granularity of the RMI Granule size.
TRACKING_NONE	Region is not tracked.
TRACKING_RESERVED	Region is reserved for use by the platform.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{XKVBY} At platform boot, the state of all tracking regions within the physical address range(s) which are reserved for DRAM is IMPLEMENTATION DEFINED.

R_{TSWBY} At platform boot, the state of all tracking regions within the physical address range(s) which are reserved for coherent device memory is IMPLEMENTATION DEFINED.

R_{PNXFP} At platform boot, the state of all tracking regions within the physical address range(s) which are reserved for device memory is IMPLEMENTATION DEFINED.

D_{YMQRF} A physical address is *tracked* if it is within a tracking region whose state is not TRACKING_NONE.

I_{TFTXL} Attributes of a Granule tracking region can be read by execution of RMI_GRANULE_TRACKING_GET.

I_{YJHVZ} Attributes of a Granule tracking region can be modified by execution of RMI_GRANULE_TRACKING_SET.

I_{YNZTP} Execution of RMI_GRANULE_TRACKING_SET fails with RMI_ERROR_GLOBAL unless the RMM state is RMM_STATE_ACTIVE.

I_{JXKKB} Changing the granularity of a tracking region for which tracking is enabled is permitted only if all Granules within the region have the same state.

I_{WZFYF} Permitted Granule tracking region state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

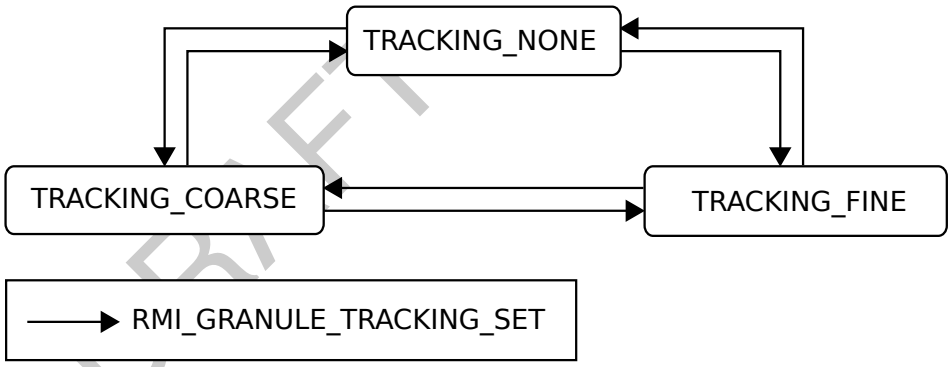


Figure A2.5: Granule tracking region state transitions

R_{CZHXN} The Granule tracking metadata which describes an address range which is populated by DRAM must be located in DRAM.

I_{XRDYF} Granule tracking metadata is permitted to be located within the tracking region that it describes.

R_{HZXDR} On execution of RMI_GRANULE_TRACKING_SET, if the implementation requires memory to be donated then the “donating memory to a Stateful RMI Operation (SRO)” flow is followed. Prior to donation, the state of the memory must be GRAN_DELEGATED.

R_{KWHDD} On execution of RMI_GRANULE_TRACKING_SET, if the implementation requires memory to be reclaimed then the “reclaiming memory from a Stateful RMI Operation (SRO)” flow is followed. Following reclamation, the state of the memory is GRAN_DELEGATED.

See also:

- A2.1 *RMM*
- A2.3.1 *Granule size*
- B3.61 *PaIsTracked function*
- B4.3.2.2 *Donating memory to an SRO*
- B4.3.2.3 *Reclaiming memory from an SRO*
- B4.5.10 *RMI_GRANULE_TRACKING_GET command*
- B4.5.11 *RMI_GRANULE_TRACKING_SET command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.3.5 Delegable physical memory

D_{ZVRXC} A physical address is *delegable* if all of the following are true:

- The address is *tracked*.
- The address is *populated*.
- The address is not within a region which is reserved for use by the platform. For example, all or part of a tracking region may be reserved for use by RMSD or by the RMM. Discovery of such reserved regions is out of scope of this specification.

I_{MMRVR} Delegable memory can be delegated by the Host for use to store RMM data, or to be mapped into a Realm.

I_{LWPPL} RMM objects can only be stored in Delegable conventional memory.

I_{ZBGJV} Non-delegable memory cannot be used to store RMM data and cannot be mapped into the Protected IPA space of a Realm.

D_{JPZBG} *Delegable device memory* is the union of Delegable non-coherent device memory and Delegable coherent device memory.

D_{CBFMJ} *Delegable memory* is the union of Delegable conventional memory and Delegable device memory.

I_{FHLPF} The following diagram summarizes the relationship between categories of delegable memory.

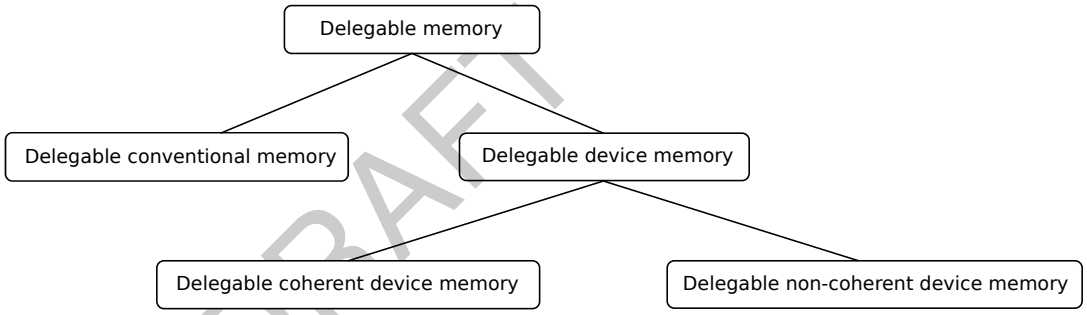


Figure A2.6: Delegable memory

DRAFT

I_{WJCMJ} The following diagram shows an example of how delegability of memory changes as a result of allocation of tracking metadata storage.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Figure A2.7: Example memory tracking view

DRAFT

See also:

- A2.3.2 *Views of physical memory*
- A2.3.4 *Granule tracking region*
- B3.52 *PaIsDelegable function*
- B3.53 *PaIsDelegableCohDevMem function*
- B3.54 *PaIsDelegableConventional function*
- B3.56 *PaIsDelegableDevMem function*
- B3.57 *PaIsDelegableNonCohDevMem function*

A2.3.6 Granule state

D_{FZNXZ} A *Granule state* indicates whether a delegable Granule has been delegated to Realm PAS, and if so whether it is in use by the RMM or by a Realm.

D_{MPLGT} The states of a Granule are listed below.

Name	Description
GRAN_DATA	Realm code or data.
GRAN_DELEGATED	Delegated for use by the RMM.
GRAN_DEV	Device memory, mapped into a Realm.
GRAN_INTERNAL	Used for IMPLEMENTATION DEFINED purposes.
GRAN_PDEV	Physical device object.
GRAN_RD	Realm Descriptor object
GRAN_REC	Realm Execution Context object.
GRAN_RTT	Realm Translation Table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
GRAN_UNDELEGATED	Not delegated for use by the RMM.
GRAN_VDEV	Virtual device object.
GRAN_VSMMU	Virtual SMMU object.

I_{MPGJV} If the state of a Granule is GRAN_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.

See also:

- A2.3.5 *Delegable physical memory*

A2.3.6.1 Granule state transitions

I_{XZNHY} The initial state of all Granules of Delegable memory is GRAN_UNDELEGATED.

I_{BWCNY} The set of reachable states depends on the Granule category.

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

DRAFT

From state	To state	Events
GRAN_UNDELEGATED	GRAN_DELEGATED	RMI_GRANULE_RANGE_DELEGATE
GRAN_DELEGATED	GRAN_UNDELEGATED	RMI_GRANULE_RANGE_UNDELEGATE
GRAN_DELEGATED	GRAN_RD	RMI_REALM_CREATE
GRAN_RD	GRAN_DELEGATED	RMI_REALM_DESTROY
GRAN_DELEGATED	GRAN_DATA	RMI_RTT_DATA_MAP_INIT RMI_RTT_DATA_MAP
GRAN_DATA	GRAN_DELEGATED	RMI_RTT_DATA_UNMAP
GRAN_DELEGATED	GRAN_REC	RMI_REC_CREATE
GRAN_REC	GRAN_DELEGATED	RMI_REC_DESTROY
GRAN_DELEGATED	GRAN_RTT	RMI_REALM_CREATE RMI_RTT_CREATE
GRAN_RTT	GRAN_DELEGATED	RMI_REALM_DESTROY RMI_RTT_DESTROY
GRAN_DELEGATED	GRAN_PDEV	RMI_PDEV_CREATE
GRAN_PDEV	GRAN_DELEGATED	RMI_PDEV_DESTROY
GRAN_DELEGATED	GRAN_VDEV	RMI_VDEV_CREATE
GRAN_VDEV	GRAN_DELEGATED	RMI_VDEV_DESTROY
GRAN_DELEGATED	GRAN_DEV	RMI_RTT_DEV_MAP
GRAN_DEV	GRAN_DELEGATED	RMI_RTT_DEV_UNMAP
GRAN_DELEGATED	GRAN_VSMMU	RMI_VSMMU_CREATE

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

From state	To state	Events
GRAN_VSMMU	GRAN_DELEGATED	RMI_VSMMU_DESTROY
GRAN_DELEGATED	GRAN_INTERNAL	RMI_OP_MEM_DONATE
GRAN_INTERNAL	GRAN_DELEGATED	RMI_OP_MEM_RECLAIM
GRAN_UNDELEGATED	GRAN_INTERNAL	RMI_OP_MEM_DONATE
GRAN_INTERNAL	GRAN_UNDELEGATED	RMI_OP_MEM_RECLAIM

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

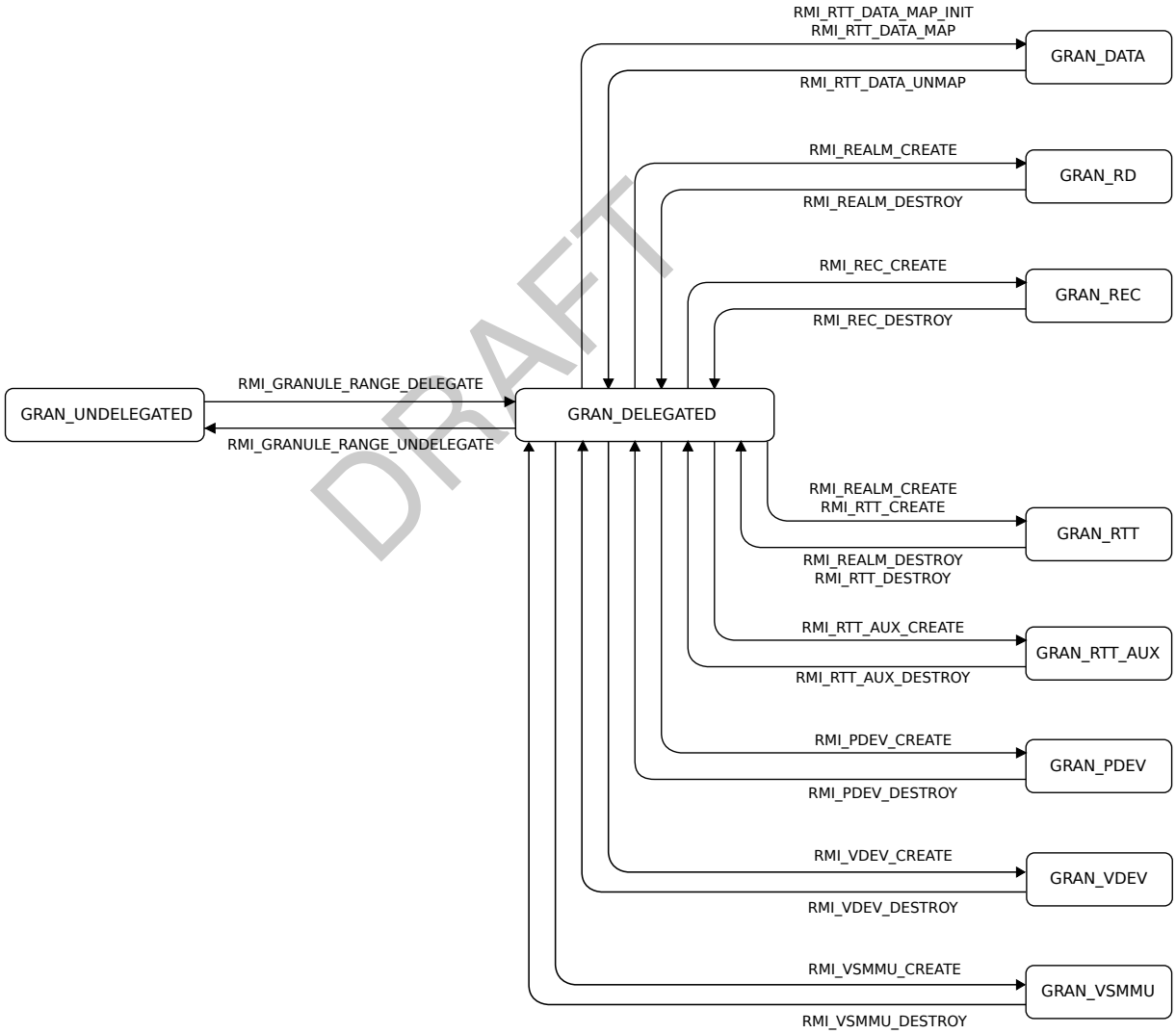


Figure A2.8: Granule state transitions for Delegable conventional memory

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Figure A2.9: Granule state transitions for Delegable device memory

See also:

- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*
- B4.5.9 *RMI_GRANULE_RANGE_UNDELEGATE command*
- B4.5.14 *RMI_OP_MEM_DONATE command*
- B4.5.15 *RMI_OP_MEM_RECLAIM command*
- B4.5.18 *RMI_PDEV_CREATE command*
- B4.5.19 *RMI_PDEV_DESTROY command*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.5.38 *RMI_REALM_DESTROY command*
- B4.5.40 *RMI_REC_CREATE command*
- B4.5.41 *RMI_REC_DESTROY command*
- B4.5.56 *RMI_RTT_CREATE command*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.61 *RMI_RTT_DEV_MAP command*
- B4.5.62 *RMI_RTT_DEV_UNMAP command*
- B4.5.74 *RMI_VDEV_CREATE command*
- B4.5.75 *RMI_VDEV_DESTROY command*
- B4.5.85 *RMI_VSMMU_CREATE command*
- B4.5.86 *RMI_VSMMU_DESTROY command*

DRAFT

A2.3.6.2 Granule delegation

D_{ZGVYY} Transition of Granules from GRAN_UNDELEGATED to GRAN_DELEGATED state is called *Granule delegation*.

D_{TRDLJ} Transition of Granules from GRAN_DELEGATED to GRAN_UNDELEGATED state is called *Granule undelegation*.

I_{WFWGV} Granule delegation and undelegation implement the *Range RMI operation which returns progress address* programming model.

I_{MVQHM} Granule delegation and undelegation implement the *RMI operations which can lead to an intermediate state* programming model.

I_{GTLRC} The amount of progress made by a Granule delegation or undelegation operation during a single RMI call is determined by a combination of constraints imposed by the RMM architecture, and IMPLEMENTATION DEFINED properties of the RMM implementation.

The following architectural constraints apply to the tracking region which includes the start of the target PA range:

- If the tracking region is untracked then the command fails with RMI_ERROR_TRACKING.
- If start of the target PA range is not aligned to the granularity of the tracking region then the command fails with RMI_ERROR_INPUT.
- If the granularity of the tracking region is larger than the total size of the target PA range then the command fails with RMI_ERROR_INPUT.

The following architectural constraints apply to the Granule at the start of the target PA range:

- If the state of the Granule is not the expected source state then the command fails with RMI_ERROR_INPUT. For example, for Granule delegation the source state is GRAN_UNDELEGATED.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

If none of the above checks failed then the operation starts to operate on the target PA range. The operation terminates when any of the following is true:

- The operation has started to transition one or more contiguous Granules from the source state, but has not completed the work needed for the entry to reach the destination state. For example, for Granule delegation the destination state is GRAN_DELEGATED. The Granules transition to an intermediate state and the command returns RMI_INCOMPLETE.
- The operation reaches a tracking region or Granule which fails one of the above checks. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made. If no changes of system state occur before the operation is resumed then the next command will fail the same check at the beginning, and return an error code as described above.
- The operation reaches a tracking region which passes the above checks, but whose size is too large to process for an IMPLEMENTATION DEFINED reason. For example, an implementation may limit the maximum size of the address space which can be processed during a single RMI command, either to ensure that the command's execution time does not exceed an IMPLEMENTATION DEFINED budget, or to limit complexity of the implementation. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made. If no changes of system state occur before the operation is resumed then the next command will fail with RMI_ERROR_TRACKING. This indicates that in order to progress the operation, the Host must first transition the tracking region to fine granularity.
- The operation neither encounters a failing check, nor reaches the end of the target PA range, but yields for an IMPLEMENTATION DEFINED reason. For example, this could be due to the command's execution time having exceeded an IMPLEMENTATION DEFINED budget. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made.
- The operation reaches the end of the target PA range. The command returns RMI_SUCCESS with `out_top == top`.

I_{LGWLL} On successful execution of a Granule delegation or undelegation command, all of the following are true:

- `out_top > base`
- The state of all Granules in the PA range `[base, out_top)` transition from the source state to the destination state.

I_{LPZRM} On failed execution of a Granule delegation or undelegation command, if the result is not RMI_INCOMPLETE then no Granules in the PA range `[base, top)` transition from the source state to the destination state.

I_{JXDYB} Execution of RMI_GRANULE_RANGE_DELEGATE fails with RMI_ERROR_GLOBAL unless the RMM state is RMM_STATE_ACTIVE.

See also:

- B4.3.2 *Stateful RMI operations*
- B4.3.5 *Range RMI operations*
- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*
- B4.5.9 *RMI_GRANULE_RANGE_UNDELEGATE command*

A2.3.7 Granule ownership

I_{DMVQM} A Granule whose state is one of the following is owned by a Realm:

- GRAN_DATA
- GRAN_DEV
- GRAN_RD
- GRAN_REC
- GRAN_RTT
- GRAN_VDEV
- GRAN_VSMMU

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 GRAN_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 GRAN_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 RTTE_TABLE. Recursively following the parent relationship leads to the RD of the owning Realm.

I_{QCNRM} A Granule whose state is GRAN_DATA is mapped at a Protected IPA, in an RTT entry whose state is RTTE_DATA. 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.

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

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

See also:

- A2.2 *Realm*
- A2.2.7 *Realm Descriptor*
- A2.4 *Realm Execution Context*
- A5.2.1 *Realm IPA space*
- A5.6 *Realm Translation Table*
- Chapter A9 *Realm device assignment*
- B4.5.40 *RMI_REC_CREATE command*
- B4.5.56 *RMI_RTT_CREATE command*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*

DRAFT

A2.3.8 Granule wiping

R_{TMGSL} When the state of a Granule has transitioned from *P* to GRAN_DELEGATED and then to any other state, any content associated with *P* has been *wiped*.

X_{CTGQZ} Any sequence of Granule state transitions which passes through the GRAN_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 GRAN_DELEGATED.

D_{WTWJR} *Wiping* is an operation which changes the observable value of a memory location from *X* to *Y*, such that the value *X* cannot be determined from the value *Y*.

R_{BSXXV} Wiping of a memory location does not reveal, directly or indirectly, any confidential Realm data.

I_{MRPCQ} Possible implementations of wiping include:

- The RMM (or other platform firmware) writing either random data or zeroes to the memory location
- The MEC of the memory location being changed
- The state of a device, which is observable via MMIO to the memory location, being reset

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_RTT_DATA_MAP) are zero.

See also:

- *Arm CCA Security model* [4]
- A2.3.6 *Granule state*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.57 *RMI_RTT_DATA_MAP command*

A2.3.9 Granule Protection Table management

D_{VJQHR} *GPT unfolding* is the operation of creating an L1GPT, by applying the following process:

1. Locate the L0GPT entry which describes the target physical address range.
2. Check that the L0GPT entry is a Block descriptor.
3. Transition the state of all Granules in the L1GPT from GRAN_UNDELEGATED to GPT_L1.
4. Populate all entries of the L1GPT with the GPI value taken from the L0GPT Block descriptor.
5. Replace the L0GPT Block descriptor with an L0GPT Table descriptor whose output address points to the L1GPT.

I_{DWPCQ} GPT unfolding is performed by execution of RMI_GPT_L1_CREATE.

D_{MZMPP} An L1GPT is *homogeneous* if all its entries have the same GPI value.

I_{PMTSD} The function `GptL1IsHomogeneous()` is used to evaluate whether an L1GPT is homogeneous.

D_{MPRKL} An L1GPT is *self-describing* if it is stored within the physical address range which it describes.

D_{DZCYR} *GPT folding* is the operation of destroying an L1GPT, by applying the following process:

1. Check that the L1GPT is homogeneous. If the L1GPT is self-describing then this check excludes the GPIs which describe the memory which stores the L1GPT itself.
2. Replace the parent L0GPT Table descriptor with an L0GPT Block descriptor, copying the GPI value from the L1GPT.
3. Transition the state of all Granules in the L1GPT from GPT_L1 to GRAN_UNDELEGATED.

I_{ZXZPC} GPT folding is performed by execution of RMI_GPT_L1_DESTROY.

R_{NYJMZ} On execution of RMI_GPT_L1_CREATE, the memory which constitutes the L1GPT is provided via the “donate memory to a Stateful RMI Operation (SRO)” flow.

I_{QJLXM} If RMI_GPT_L1_CREATE fails then the Granules which constitute the L1GPT remain in the Non-secure PAS.

I_{WSZDT} If RMI_GPT_L1_CREATE returns RMI_INCOMPLETE then the creation operation must be completed before the L1GPT can be destroyed.

R_{WGJGX} On execution of RMI_GPT_L1_DESTROY, the memory which constitutes the L1GPT is returned via the “reclaim memory from a Stateful RMI Operation (SRO)” flow.

I_{LBWFT} If RMI_GPT_L1_DESTROY returns RMI_INCOMPLETE then the destruction operation must be completed before a new L1GPT which describes the same physical address region can be created.

S_{QQDMX} Creation or destruction of L1GPTs which describes different physical address regions may be performed concurrently.

S_{PLQRJ} RMI_GPT_L1_CREATE allows the Host to utilize NUMA awareness when allocating an L1GPT.

See also:

- A5.6.6 *RTT folding*
- A5.6.7 *RTT unfolding*
- B3.33 *GptL1IsHomogeneous function*
- B4.3.2.2 *Donating memory to an SRO*
- B4.3.2.3 *Reclaiming memory from an SRO*
- B4.5.6 *RMI_GPT_L1_CREATE command*
- B4.5.7 *RMI_GPT_L1_DESTROY command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A2.4 Realm Execution Context

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

A2.4.1 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:

- A2.2.2 *Realm execution environment*
- Chapter A4 *Realm exception model*

A2.4.2 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
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 RIPAS_DESTROYED to RIPAS_RAM should be permitted
ripas_response	RmmRecResponse	Host response to RIPAS change request
dev_mem_addr	Address	Next IPA to be processed in VDEV mapping validation

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
dev_mem_top	Address	Top IPA of pending VDEV mapping validation
dev_mem_pa	Address	PA of device memory
dev_mem_flags	RmmDevMemFlags	VDEV mapping validation flags
dev_mem_response	RmmRecResponse	Host response to VDEV 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_1	Bits64	Virtual device ID 1
vdev_pa_1	Address	VDEV PA
vdev_id_2	Bits64	Virtual device ID 2
vdev_attest_info_1	RmmVdevAttestInfo	Attestation information for first VDEV
vdev_attest_info_2	RmmVdevAttestInfo	Attestation information for second VDEV
vsmmu_vsid	Bits64	Virtual SMMU Stream ID
vsmmu_resp	Bits64	PRI response
vsmmu_pasid	Bits64	PASID

DRAFT

I_{BGVRT} The *gprs* 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 *mpidr* attribute of a REC is a value which can be used to identify the VPE associated with the REC.

I_{BLVKZ} The *pc* 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 *runnable* 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 *state* attribute of a REC is controlled by the Host, by execution of the RMI_REC_ENTER command.

D_{CDXDZ} The *sysregs* 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_{RWKGW} The *gic_owner* attribute of a REC is the index of the Plane which is the GIC owner for the REC.

See also:

- A2.4.3 *REC index and MPIDR value*
- A2.4.4 *REC lifecycle*
- A4.3.4.3 *REC exit due to Data Abort*
- A10.4 *Planes interrupts*
- B4.5.42 *RMI_REC_ENTER command*
- B6.3.2 *PSCI_CPU_OFF command*
- B6.3.3 *PSCI_CPU_ON command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- C2.50 *RmmRec type*

A2.4.3 REC index and MPIDR value

D_{KQVHN} The *REC index* is the unsigned integer value generated from MPIDR fields as follows:

```
index = aff0
  + 16 * aff1
  + 16 * 256 * aff2
  + 16 * 256 * 256 * aff3
```

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
…	…	…	…	…

DRAFT

I_{PVLZY} The `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:

- B3.102 *RecIndex function*
- B4.5.40 *RMI_REC_CREATE command*
- B4.6.68 *RmiRecMpidr type*

A2.4.4 REC lifecycle

A2.4.4.1 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.

A2.4.4.2 State transitions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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
NULL	REC_READY	RMI_REC_CREATE
REC_READY	*NULL*	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.

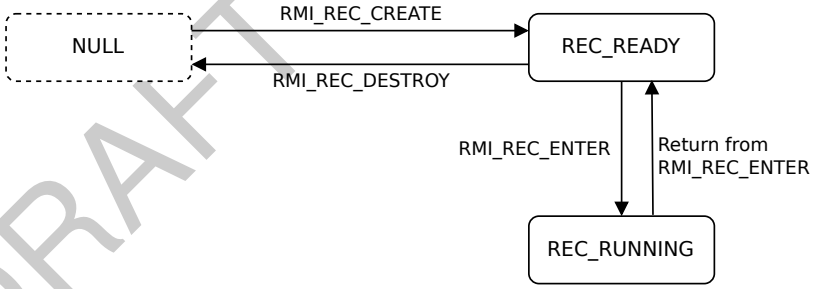


Figure A2.10: REC state transitions

DRAFT

I_LYXCN The maximum number of RECs per Realm is an IMPLEMENTATION DEFINED value which is discoverable via RMI_FEATURES.

See also:

- B4.5.40 *RMI_REC_CREATE command*
- B4.5.41 *RMI_REC_DESTROY command*
- B4.5.42 *RMI_REC_ENTER command*

See also:

- B4.5.5 *RMI_FEATURES command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A3
Feature discovery and configuration

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 *pseudo-register* 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:

- A2.2.6 *Realm parameters*
- A7.1.1 *Realm Initial Measurement*
- B3.96 *RealmParamsSupported function*
- B4.5.5 *RMI_FEATURES command*
- B4.5.37 *RMI_REALM_CREATE command*
- B5.4.4 *RSI_FEATURES command*
- C2.16 *RmmGlobalStatic type*

DRAFT

A3.2 Realm hash algorithm

I_{WMKGX} The set of hash algorithms supported by the implementation is reported by the RMI_FEATURES command in RmiFeatureRegister1.

I_{JKZBC} 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*
- B3.96 *RealmParamsSupported function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.24 *RmiFeatureRegister1 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::LPA2.

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

X_{FTVXQ} The Host may want to enable LPA2 for a Realm due to either or both of the following reasons:

- 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} 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:

- RMI_RTT_DATA_MAP_INIT
- RMI_RTT_DATA_MAP
- RMI_RTT_CREATE
- RMI_RTT_AUX_CREATE
- RMI_RTT_UNPROT_MAP

See also:

- A5.2.1 *Realm IPA space*
- B3.96 *RealmParamsSupported function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.23 *RmiFeatureRegister0 type*
- B5.4.16 *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 RmiFeatureRegister1::SVE.

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:

- B3.96 *RealmParamsSupported function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.24 *RmiFeatureRegister1 type*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A3.5 Realm support for self-hosted debug

I_{SSTJD} Self-hosted debug is always available in Armv8-A.

I_{VYXGY} The number of breakpoints and watchpoints available are reported by the RMI_FEATURES command in RmiFeatureRegister0::{NUM_BPS,NUM_WPS}.

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:

- B3.96 *RealmParamsSupported function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.23 *RmiFeatureRegister0 type*

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::PMU.

I_{NHCFC} 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_{NTWXF} 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*
- B3.96 *RealmParamsSupported function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.23 *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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

A3.9 Realm support for Trace Buffer Extension

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

A3.10 Support for Realm device assignment

I_{ZTHZG} Support by the implementation for Realm device assignment is reported by the RMI_FEATURES command in RmiFeatureRegister2::DA.

I_{SYPHS} 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*
- B3.96 *RealmParamsSupported function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.25 *RmiFeatureRegister2 type*

A3.11 Support for auxiliary Planes

I_{TMWPW} The maximum number of auxiliary Planes supported by the implementation is reported by the RMI_FEATURES command in of RmiFeatureRegister3::MAX_NUM_AUX_PLANED.

R_{CHSMV} The maximum number of auxiliary Planes supported by the implementation is either 0 or 3.

I_{HLGGD} The number of auxiliary Planes for a Realm is provided by the Host when calling RMI_REALM_CREATE.

R_{GTVWF} Providing a number of auxiliary Planes which is larger than the maximum number of auxiliary Planes causes RMI_REALM_CREATE to fail.

I_{SQVMF} For a Realm with a non-zero number of auxiliary Planes, the RmiFeatureRegister3::RTT_PLANE field 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.

I_{YPTCP} Whether a Realm has an RTT tree per Plane is configured by the Host when calling RMI_REALM_CREATE.

See also:

- Chapter A10 *Planes*
- B3.96 *RealmParamsSupported function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.26 *RmiFeatureRegister3 type*
- C2.16 *RmmGlobalStatic type*

A3.12 Support for Stage 2 Access Permissions indirect encoding

I_{QFKTM} The RmiFeatureRegister3::RTT_S2AP_INDIRECT field 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].

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{PSWPG} 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*
- B3.96 *RealmParamsSupported function*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.26 *RmiFeatureRegister3 type*
- C2.16 *RmmGlobalStatic type*

A3.13 Live Firmware Activation

I_{JQTKZ} 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.

I_{LKFNN} A Realm has an LFA policy which is provided by the Host when calling RMI_REALM_CREATE.

R_{VGKGM} 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.

I_{BJSZP} 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.

I_{DVKZX} 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.

I_{DNMVM} 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.

I_{HVGSH} The LFA policy of a Realm is reflected in the Realm attestation token.

See also:

- *Live Firmware Activation SMC Interface* [5]
- A7.2.3.3.8 *CCA platform software components claim*
- B4.5.37 *RMI_REALM_CREATE command*
- B4.6.34 *RmiLfaPolicy type*

DRAFT

A3.14 Support for Realm memory encryption

I_{DPVMW} A Realm has an *MEC policy* which is provided by the Host when calling RMI_REALM_CREATE.

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*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A4
Realm exception model

DRAFT

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

See also:

- A2.2.2 *Realm execution environment*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A4.1 Realm exception model overview

D_{HCGWL} A *Realm entry* is a transfer of control to a Realm.

D_{RMGWJ} A *Realm exit* 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 *REC entry* 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 `rec` and "the target REC" refer to the REC object which is provided to the RMI_REC_ENTER command.

D_{BLJGY} A *RecRun object* 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 with the target REC having made no forward progress. For example, on observing a pending interrupt, the implementation can generate a REC exit due to IRQ without the target REC having executed any instructions.

D_{TJCGH} A *REC exit* 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.

DRAFT

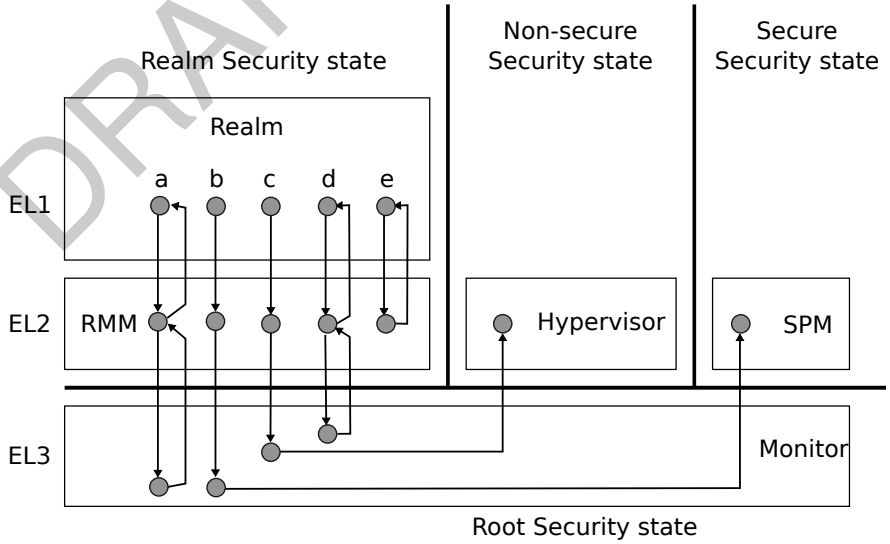


Figure A4.1: Realm exit paths

a. The exception is taken to EL2. The RMM decides to perform a REC exit. The RMM executes an SMC instruction, yielding control to the Monitor. The Monitor handles the exception and returns control to the Realm.

b. The exception is taken to EL2. The RMM decides to perform a REC exit. The RMM executes an SMC instruction, yielding control to the Monitor. The Monitor passes control to the Secure Security state. This may be for example due to an FIQ.

c. 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

d. 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.

e. 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*
- B4.5.42 *RMI_REC_ENTER command*
- B4.6.70 *RmiRecRun type*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A4.2 REC entry

This section describes REC entry.

See also:

- A4.3 *REC exit*
- B4.5.42 *RMI_REC_ENTER command*

A4.2.1 RmiRecEnter object

D_{SVSJM} An *RmiRecEnter object* is a data structure used to pass values from the Host to the RMM on REC entry.

I_{YSKDN} An RmiRecEnter object is stored in the RecRun object which is passed by the Host as an input to the RMI_REC_ENTER command.

I_{TRKKX} On REC entry, execution state is restored from the REC object and from the RmiRecEnter object to the PE.

I_{GHDLM} An RmiRecEnter object contains attributes which are used to manage Realm virtual interrupts.

D_{CLNLW} The attributes of an RmiRecEnter object are summarized in the following table.

Name	Byte offset	Type	Description
flags	`0x0`	RmiRecEnterFlags	Flags
gprs[31]	`0x200`	Bits64	Registers

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:

- A2.4 *Realm Execution Context*
- A4.3.1 *RmiRecExit object*
- Chapter A6 *Realm interrupts and timers*
- B4.6.64 *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:

- 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 `rec.pending` is REC_PENDING_HOST_CALL, then GPR values X0 to X30 are copied from `rec_enter.gprs[0..30]` to the RsiHostCall data structure.

R_{YWHKC} On REC entry, if writing to the RsiHostCall 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 RsiHostCall data structure succeeds then `rec.pending` is REC_PENDING_NONE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A4.3 REC exit

This section describes REC exit.

See also:

- A4.2 *REC entry*
- B4.5.42 *RMI_REC_ENTER command*

A4.3.1 RmiRecExit object

D_{PBDCB} An *RmiRecExit object* is a data structure used to pass values from the RMM to the Host on REC exit.

I_{VHJTL} An RmiRecExit object is stored in the RecRun object which is passed by the Host as an input to the RMI_REC_ENTER command.

I_{JKWPB} On REC exit, execution state is saved from the PE to the REC object and to the RmiRecExit object.

I_{ZSCNM} An RmiRecExit object contains attributes which are used to manage Realm virtual interrupts and Realm timers.

D_{FFCMN} The attributes of an RmiRecExit 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
gprs[31]	0x200	Bits64	Registers
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_1	0x530	Bits64	Virtual device ID 1
vdev_id_2	0x538	Bits64	Virtual device ID 2

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
dev_mem_base	0x540	Bits64	Base IPA of target region for VDEV mapping validation
dev_mem_top	0x548	Bits64	Top IPA of target region for VDEV mapping validation
dev_mem_pa	0x550	Address	Base PA of device memory region
imm	0x600	Bits16	Host call immediate value
plane	0x608	UInt64	Plane index
pmu_ovf_status	0x700	RmiPmuOverflowStatus	PMU overflow status
vsmmu	0x710	Address	Virtual SMMU base IPA

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:

- A2.4 *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*
- B4.6.66 *RmiRecExit type*

DRAFT

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:

- B4.6.67 *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 except for the following, `rec_exit.gprs` is zero.

- REC exit due to Host call
- REC exit due to PSCI
- REC exit due to Emulatable Data Abort, and the Realm memory access was a write

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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	If taken from the Primary Plane, REC exit due to WFI or WFE Otherwise, Plane exit due to synchronous exception.
HVC instruction execution in AArch64 state	If taken from the Primary Plane, Unknown exception taken to Realm. Otherwise, Plane exit due to synchronous exception.
SMC instruction execution in AArch64 state	If taken from the Primary Plane, one of: • SMCCC_VERSION emuulated by RMM • REC exit due to PSCI • RSI command handled by RMM, followed by return to Realm Otherwise, Plane exit due to synchronous exception.
Trapped MSR, MRS or System instruction execution in AArch64 state	If taken from the Primary Plane, one of: • REC exit due to System register access • Emulated by RMM, followed by return to Realm Otherwise, Plane exit due to synchronous exception.
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

DRAFT

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:

- SMCCC_VERSION
- PSCI
- RSI

See also:

- A4.5 *Host call*
- A10.2.3 *Plane exit*
- 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.RN` is zero.
- If `rec_exit.esr.ISS.TI[1] == 1` then `rec_exit.esr.ISS.RV == 1` otherwise `rec_exit.esr.ISS.RV == 0`.
- `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.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.gprs[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 any of the following is true:

- HIPAS_VOID and RIPAS_RAM
- RIPAS_DESTROYED
- The access is from a Plane which uses an Auxiliary RTT tree, and the IPA is not mapped in that tree

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.cnt*` and `rec_exit.pmu_ovf_status` are zero.

I_{HMFFM} Low bits of `HPFAR_EL2.FIPA` are set to zero in order to align the address to the RMI Granule size. `rec_exit.hpfar` therefore only reveals the Realm's access patterns at the granularity of the RMI Granule size.

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*
- A10.3.1 *Auxiliary RTT*

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 which is HIPAS_UNMAPPED_NS, where the access caused `ESR_EL2.ISS.ISV` to be set to `'1'`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- an Unprotected IPA which is HIPAS_MAPPED_NS, where the access caused a stage 2 permission fault and caused `ESR_EL2.ISS.ISV` to be set to `'1'`
- the access is from a Plane which uses an Auxiliary RTT tree, and the IPA is not mapped in that tree

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 which is HIPAS_UNMAPPED_NS, where the access caused `ESR_EL2.ISS.ISV` to be set to `'0'`
- an Unprotected IPA which is HIPAS_MAPPED_NS, where the access caused a stage 2 permission fault and caused `ESR_EL2.ISS.ISV` to be set to `'0'`
- a Protected IPA which is HIPAS_VOID and RIPAS_RAM
- a Protected IPA which is RIPAS_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.SRT` is zero.
- `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.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.

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_{QBTPR} 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*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A5.2.7 *Realm access to an Unprotected IPA*
- A6.1 *Realm interrupts*
- A6.2 *Realm timers*
- A8.1 *Realm PMU*
- A10.3.1 *Auxiliary RTT*

A4.3.4.4 REC exit due to System register access

D_{NNTHM} A *REC exit due to System register access* is a REC exit due to a system register access.

R_{MWZSY} On REC exit due to System register access, all of the following are true:

- `rec_exit.exit_reason` is RMI_EXIT_SYNC.
- `rec_exit.esr` contains the value of `ESR_EL2` at the time of the Realm exit.
- All other `rec_exit` fields except for `rec_exit.cnt*` and `rec_exit.pmu_ovf_status` are zero.

See also:

- A6.1 *Realm interrupts*

A4.3.5 REC exit due to IRQ

D_{YLWXK} 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 RMI_PSCI_COMPLETE
PSCI_VERSION	No	-

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

PSCI function	Can result in REC exit due to PSCI	Requires Host to call RMI_PSCI_COMPLETE
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.gprs` 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.

DRAFT

See also:

- A4.3.3 *General purpose registers saved on REC exit*
- B3.81 *PsciReturnCodePermitted function*
- B4.5.28 *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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- A2.4.2 *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.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*
- B5.4.5 *RSI_HOST_CALL command*

DRAFT

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.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

D_{YMPNX} A *REC exit due to S2AP change pending* is a REC exit due to the Realm issuing an *S2AP change request*.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{NJDXX} 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.
- All other `rec_exit` fields except for `rec_exit.cnt*` and `rec_exit.pmu_ovf_status` are zero.

I_{RNVGH} 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.

R_{XYYSY} 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:

- A2.4.2 *REC attributes*
- A10.3.2.3 *Stage 2 Access Permissions change within a multi-Plane Realm*

DRAFT

A4.3.12 REC exit due to VDEV request

D_{MXVJJ} 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.

R_{DKTFP} On REC exit due to VDEV request, `rec_exit.exit_reason` is RMI_EXIT_VDEV_REQUEST.

R_{YPMKY} On REC exit due to VDEV request, `rec_exit.vdev_id_1` contains the requested virtual device ID.

R_{RFKKD} On REC exit due to VDEV request, `rec.vdev_pending` is set to REC_PENDING_VDEV_REQUEST.

I_{SGJTP} 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:

- A9.4.4 *Mapping from virtual device ID to VDEV object*
- B4.5.73 *RMI_VDEV_COMPLETE command*

A4.3.13 REC exit due to VDEV mapping validation

D_{JMGWV} A *REC exit due to VDEV mapping validation* is a REC exit due to the Realm issuing a *VDEV mapping validation request*.

R_{TCFTR} On REC exit due to VDEV mapping validation, all of the following are true:

- `rec_exit.exit_reason` is RMI_EXIT_VDEV_VALIDATE_MAPPING.
- `rec_exit.vdev_id_1` is the virtual device identifier.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- `rec_exit.dev_mem_base` is the base IPA of the region on which VDEV mapping validation is pending.
- `rec_exit.dev_mem_top` is the top IPA of the region on which VDEV mapping validation is pending.
- `rec.dev_mem_addr` is the base IPA of the region on which VDEV mapping validation is pending.
- `rec.dev_mem_top` is the top IPA of the region on which VDEV 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.
- All other `rec_exit` fields except for `rec_exit.cnt*` and `rec_exit.pmu_ovf_status` are zero.

I_{JTQLP} On REC exit due to VDEV mapping validation:

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

See also:

- A2.4.2 *REC attributes*
- A5.5 *VDEV mapping validation*
- A9.6.2 *Realm validation of device memory mappings*
- B4.5.63 *RMI_RTT_DEV_VALIDATE command*

DRAFT

A4.3.14 REC exit due to VSMMU command

D_{DZYKY} A *REC exit due to VSMMU command* is a REC exit due to the Realm enqueing a VSMMU command which requires action to be taken by the Host.

R_{CXSBP} On REC exit due to VSMMU command, all of the following are true:

- `rec_exit.exit_reason` is RMI_EXIT_VSMMU_COMMAND.
- `rec_exit.vsmmu` is the Virtual SMMU base IPA for which the command was enqueued.
- All other `rec_exit` fields except for `rec_exit.cnt*` and `rec_exit.pmu_ovf_status` are zero.

I_{DDMRF} Following REC exit due to VSMMU command, the Host is expected to call the RMI_VSMMU_CMD_GET command. The return values of that command include flags which inform the Host which further action is required.

See also:

- A2.4.2 *REC attributes*
- A9.8.5 *VSMMU commands*
- B4.5.84 *RMI_VSMMU_CMD_GET command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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*
- B5.4.5 *RSI_HOST_CALL command*
- D1.3.2 *Host call flow*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A5
Realm memory management

DRAFT

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:

- A2.2.2 *Realm execution environment*
- D1.5 *Realm memory management flows*
- Chapter D2 *Realm shared memory protocol*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- A2.2.3 *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
RIPAS_DESTROYED	Address which is inaccessible to the Realm due to an action taken by the Host.
RIPAS_DEV	Address where memory of an assigned Realm device is mapped.
RIPAS_EMPTY	Address where no Realm resources are mapped.
RIPAS_RAM	Address where private code or data owned by the Realm is mapped.

I_{VZCZV} RIPAS values are stored in leaf entries of the primary RTT.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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*
- B5.4.6 *RSI_IPA_STATE_GET command*

A5.2.3 Realm access to a Protected IPA

R_{JVQQR} Realm data access to a Protected IPA which is RIPAS_EMPTY causes a Synchronous External Abort taken to the Realm.

R_{MKLSD} Realm instruction fetch from a Protected IPA which is RIPAS_EMPTY causes a Synchronous External Abort taken to the Realm.

I_{QSQLF} Realm data access to a Protected IPA which is RIPAS_RAM may cause a Synchronous External Abort taken to the Realm, if the access results in a Synchronous External Abort which is restartable or recoverable.

I_{PGHBT} Realm data access to a Protected IPA which is RIPAS_RAM can cause an REC exit due to Data Abort.

I_{FCJCP} Realm instruction fetch from a Protected IPA which is RIPAS_RAM may cause a Synchronous External Abort taken to the Realm, if the access results in a Synchronous External Abort which is restartable or recoverable.

I_{XHKQY} Realm instruction fetch from a Protected IPA which is RIPAS_RAM can cause a REC exit due to Instruction Abort.

R_{CLVKF} Realm data access to a Protected IPA which is RIPAS_DESTROYED causes a REC exit due to Data Abort.

R_{MZYQT} Realm instruction fetch from a Protected IPA which is RIPAS_DESTROYED causes a REC exit due to Instruction Abort.

I_{LSNTT} Realm data access to a Protected IPA which is RIPAS_DEV may cause a Synchronous External Abort taken to the Realm, if the access results in a Synchronous External Abort which is restartable or recoverable.

I_{XDXVV} Realm data access to a Protected IPA which is RIPAS_DEV can cause an REC exit due to Data Abort.

R_{ZQTLP} Realm instruction fetch from a Protected IPA which is RIPAS_DEV causes a Synchronous External Abort taken to the Realm.

DRAFT

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*
- A9.9.1 *Device access to a Protected IPA*

A5.2.4 RSI command access to a Protected IPA

R_{MXTQZ} Access by an RSI command to a Protected IPA which is RIPAS_EMPTY causes the command to fail and return an error to the Realm.

R_{PKTCR} Access by an RSI command to a Protected IPA whose RIPAS is not RIPAS_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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{BSBHN} For a Realm in the REALM_NEW state, the RIPAS of a Protected IPA can change to RIPAS_RAM due to Host execution of RMI_RTT_DATA_MAP_INIT or RMI_RTT_INIT_RIPAS.

I_{BSGSW} For a Realm in the REALM_NEW state, changing the RIPAS of a Protected IPA to RIPAS_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 RIPAS_DESTROYED due to Host execution of RMI_RTT_DATA_UNMAP or RMI_RTT_DESTROY.

I_{YXLCP} For a Realm in the REALM_NEW state, changing the RIPAS of a Protected IPA to RIPAS_DESTROYED does not cause the RIM to be updated.

See also:

- A5.4 *RIPAS change*
- A7.1.1 *Realm Initial Measurement*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.65 *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_{NZXPG} A Realm in the REALM_ACTIVE state can request the RIPAS of a region of Protected IPA space to be changed to RIPAS_EMPTY, RIPAS_RAM or RIPAS_DEV.

I_{RXHXF} A Realm in the REALM_ACTIVE state cannot request the RIPAS of a region of Protected IPA space to be changed to RIPAS_DESTROYED.

I_{FRJJH} For a Realm in the REALM_ACTIVE state, the RIPAS of a Protected IPA can change to RIPAS_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 RIPAS_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 RIPAS_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 RIPAS_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 RIPAS_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_RTT_DATA_MAP_INIT, establishing a mapping to physical memory, changing RIPAS to RIPAS_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 RIPAS_EMPTY.

3. Realm executes RSI_IPA_STATE_SET(ripas=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 RIPAS_RAM.

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

I_{LXNWJ} For a Realm in the REALM_ACTIVE state, the RIPAS of a Protected IPA can change to RIPAS_DEV only in response to Realm execution of RSI_VDEV_VALIDATE_MAPPING.

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

X_{JJPHJ} The result of changing the RIPAS of a Protected IPA to RIPAS_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.

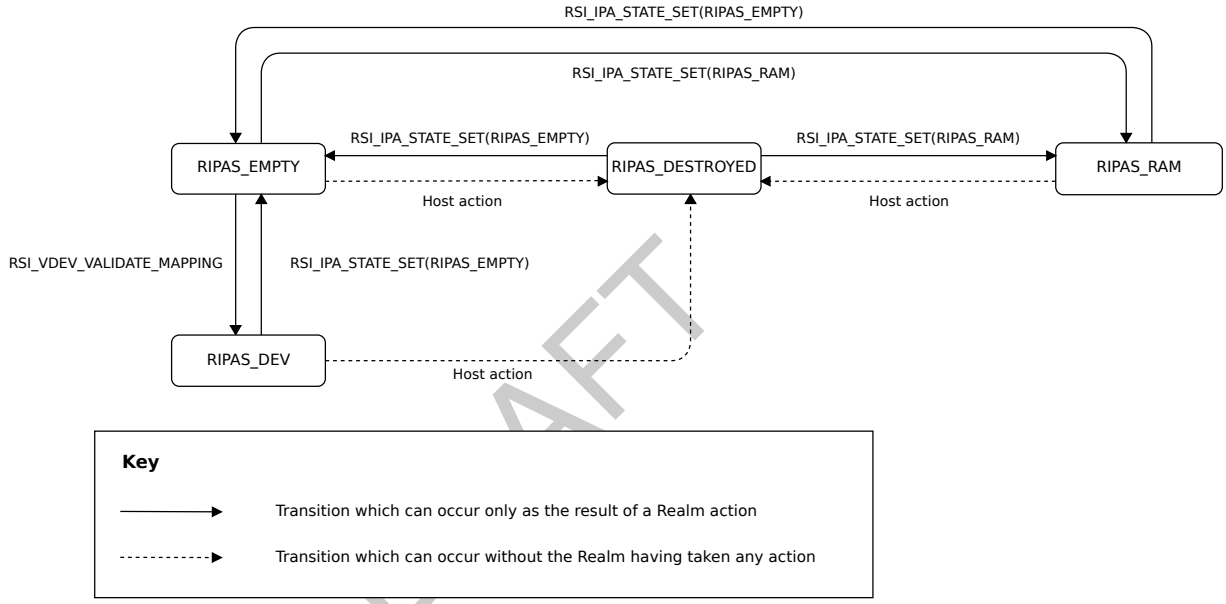


Figure A5.1: RIPAS changes which can occur when the Realm state is REALM_ACTIVE

See also:

- A5.4 *RIPAS change*
- A5.5 *VDEV mapping validation*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.63 *RMI_RTT_DEV_VALIDATE command*
- B4.5.65 *RMI_RTT_INIT_RIPAS command*
- B5.4.7 *RSI_IPA_STATE_SET command*
- B5.4.20 *RSI_VDEV_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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 `ESR_EL2.ISS.ISV` 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:

- 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, `ESR_EL1.EA == '1'`.

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A5.2.10 Summary of Realm IPA space properties

I_{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	When access results in an SEA which is recoverable or restartable	When HIPAS is not HIPAS_DATA	When access results in an SEA which is recoverable or restartable	When HIPAS is not HIPAS_DATA
Protected, RIPAS_DEV	When access results in an SEA which is recoverable or restartable	When HIPAS is none of: • HIPAS_ARCH_DEV • HIPAS_NARCH_DEV	Always (SEA)	Never
Protected, RIPAS_DESTROYED	Never	Always	Never	Always
Unprotected	Host can inject SEA following REC exit due to Data Abort. GPF occurs if NS access is not permitted by GPT entry.	When access caused a stage 2 permission fault	Always (SEA)	Never
Outside Realm IPA space	Always (Address Size Fault)	Never	Always (Address Size Fault)	Never

DRAFT

See also:

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

A5.2.11 Cache maintenance operations

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

X_{XZRDW} This is to ensure that a Realm cannot invalidate a cache line owned by another Realm.

U_{VQMTB} 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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_ARCH_DEV	Protected IPA which is associated with an architectural device.
HIPAS_DATA	Protected IPA which is associated with a DATA Granule.
HIPAS_MAPPED_NS	Unprotected IPA which is associated with a physical Granule.
HIPAS_NARCH_DEV	Protected IPA which is associated with a GRAN_DEV Granule.
HIPAS_UNMAPPED_NS	Unprotected IPA which is not associated with any Granule.
HIPAS_VOID	Protected IPA which is not associated with any Granule.

I_{TRSKJ} HIPAS values are stored in leaf entries of a primary 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 HIPAS_DATA and RIPAS_RAM.

I_{MKNDK} A mapping at a Protected IPA is valid if HIPAS_NARCH_DEV and RIPAS_DEV.

I_{STBRZ} The RMM emulates access to a Protected IPA if HIPAS_ARCH_DEV and RIPAS_DEV.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{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.

The TTD.VALID and TTD.NS columns refer to the value of the corresponding fields in the architecturally-defined Stage 2 translation table descriptor which is written by the RMM.

RIPAS	HIPAS	TTD.VALID	TTD.NS	Data access	Instruction fetch
EMPTY	VOID	0		SEA to Realm	SEA to Realm
EMPTY	DATA	0		SEA to Realm	SEA to Realm
EMPTY	NARCH_DEV	0		SEA to Realm	SEA to Realm
EMPTY	ARCH_DEV	0		SEA to Realm	SEA to Realm
RAM	VOID	0		REC exit due to Data Abort	REC exit due to Instruction Abort
RAM	DATA	1	0	Data access	Instruction fetch
RAM	NARCH_DEV	0		REC exit due to Data Abort	REC exit due to Instruction Abort
RAM	ARCH_DEV	0		REC exit due to Data Abort	REC exit due to Instruction Abort
DESTROYED	VOID	0		REC exit due to Data Abort	REC exit due to Instruction Abort
DESTROYED	DATA	0		REC exit due to Data Abort	REC exit due to Instruction Abort
DESTROYED	NARCH_DEV	0		REC exit due to Data Abort	REC exit due to Instruction Abort
DESTROYED	ARCH_DEV	0		REC exit due to Data Abort	REC exit due to Instruction Abort
DEV	NARCH_DEV	1	0	Device access	SEA to Realm
DEV	ARCH_DEV	0		Emulated by RMM	SEA to Realm

DRAFT

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*
- A9.8 *Virtual SMMU*

A5.3.2 Memory mapping operations overview

I_{HSFHR} The set of memory mapping and unmapping operations which can be performed by execution of RMI commands is listed below.

- Create or remove mappings within a contiguous range of Protected IPA space to conventional memory which is contiguous in Realm PAS.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- Create or remove mappings within a contiguous range of Protected IPA space to conventional memory which is non-contiguous in Realm PAS.
- Create or remove mappings within a contiguous range of Protected IPA space to device memory which is contiguous in Realm PAS.
- Create or remove mappings within a contiguous range of Protected IPA space to device memory which is non-contiguous in Realm PAS.
- Create or remove mappings within a contiguous range of Protected IPA space to device memory which is emulated by the RMM.
- Create or remove mappings within a contiguous range of Unprotected IPA space to conventional memory which is contiguous in Non-secure PAS.
- Create or remove mappings within a contiguous range of Unprotected IPA space to conventional memory which is non-contiguous in Non-secure PAS.
- Propagate mappings within a contiguous range of IPA space from the primary RTT tree to an auxiliary RTT tree.
- Remove mappings within a contiguous range of IPA space from an auxiliary RTT tree.

See also:

- A5.2.1 *Realm IPA space*
- A10.3.1 *Auxiliary RTT*

A5.3.3 Range-based memory operations

D_{MPJJT} A *Range-based memory operation* is an operation which creates or removes mappings within a contiguous range of IPA space.

I_{KQYXN} A Range-based memory operation implements the *Range RMI operation which returns progress address* programming model.

I_{WWGNX} A Range-based memory operation implements the *RMI operations which can lead to an intermediate state* programming model.

See also:

- B4.3.2 *Stateful RMI operations*
- B4.3.5 *Range RMI operations*

A5.3.3.1 Output address set

D_{PMBDL} An *output address set* is a set of physical addresses which are (or will be, following a mapping operation) stored in the output address field of a set of RTT entries.

I_{VPLYJ} The output addresses which are operated on by a Range-based memory operation identify one of the following:

- Physical resources, such as DRAM or device memory, which are mapped into the IPA space of the Realm
- An architectural device which is managed by the RMM.

I_{SZBBJ} When Range-based memory operation maps or unmaps physical resources, it operates on an *output address set* which is one of the following:

- A single contiguous PA range.
- A list of PA ranges.

I_{QRTQC} To perform a Range-based memory mapping operation which operates on an output address set, the Host provides to the RMM a 64-bit value and a flag. The flag specifies whether the 64-bit value is:

- A single RMI Address Range Descriptor, provided directly in a register, or
- The base address of an RMI Address Range List which is stored in Non-secure Granules. In this case, the total size of the output address set described by the list entries must be equal to the size of the target IPA range. The caller specifies the number of entries in the list.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{LZLVV} When performing a Range-based memory unmapping operation which operates on an output address set, the Host provides to the RMM a flag which specifies whether the output address set should be:

- Returned as a single RMI Address Range Descriptor, provided directly in a register, or
- Returned by populating an RMI Address Range List in Non-secure Granules provided by the Host. In this case, the caller specifies the number of entries in the list, which are available for the RMM to populate with unmapped output addresses.
- Not returned.

X_{VKKQN} The Host may optionally maintain a shadow copy of the Realm's IPA to PA mappings. In this case, the Host does not required the output address set to be returned on unmapping.

See also:

- B4.4 *RMI address range*

A5.3.3.2 Range-based memory mapping operation

I_{LPPYG} The amount of progress made on a Range-based memory mapping operation during a single RMI call is determined by a combination of constraints imposed by the RMM architecture, and IMPLEMENTATION DEFINED properties of the RMM implementation.

The following architectural constraints apply to the RTTE at the base of the target IPA range:

- If the state of the RTTE is not RTTE_VOID then the command fails with (RMI_ERROR_RTT, walk.level).
- If base of the target IPA range is not aligned to the size of the IPA space described by the RTTE then the command fails with (RMI_ERROR_RTT, walk.level).
- If the size of the IPA space described by the RTTE is larger than the total size of the target IPA range then the command fails with (RMI_ERROR_RTT, walk.level).

If the operation uses an output address set then the following architectural constraints apply to the tracking region which includes the start of the set:

- If the tracking region is untracked then the command fails with RMI_ERROR_TRACKING.
- If start of the output address set is not aligned to the granularity of the tracking region then the command fails with (RMI_ERROR_RTT, walk.level).
- If the granularity of the tracking region is larger than the total size of the target IPA range then the command fails with RMI_ERROR_TRACKING.

If the operation uses an output address set then the following architectural constraints apply to the Granule at the start of the set:

- If the state of the Granule is not GRAN_DELEGATED then the command fails with RMI_ERROR_INPUT.

If the operation uses an output address set and ATS is enabled for the target Realm then the following architectural constraints apply to the DPT entry which includes the start of the set:

- If the Level 0 DPT is in an intermediate state then the command fails with RMI_BLOCKED.
- If the Level 0 DPT has not been created then the command fails with RMI_ERROR_DPT.
- If the DPT entry is in an intermediate state then the command fails with RMI_BLOCKED.
- If start of the output address set is not aligned to the size of the PA space described by the DPT entry then the command fails with RMI_ERROR_DPT.
- If the size of the PA space described by the DPT entry is larger than the total size of the target IPA range then the command fails with (RMI_ERROR_RTT, walk.level).

If none of the above checks failed then the operation starts to operate on the target IPA range. The operation terminates when any of the following is true:

- The operation has started to transition an RTTE entry from RTTE_VOID, but has not completed the work needed for the entry to reach the target state. The RTTE entry transitions to an intermediate state and the command returns RMI_INCOMPLETE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- The operation reaches an RTTE, tracking region, Granule or DPT entry which fails one of the above checks. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made. If no changes of system state occur before the operation is resumed then the next command will fail the same check at the beginning, and return an error code as described above. This indicates that in order to progress the operation, the Host must first unfold the RTT, transition the tracking region to fine granularity, or create a Level 1 DPT.
- The operation reaches an RTTE, tracking region or DPT entry which passes the above checks, but whose size is too large to process for an IMPLEMENTATION DEFINED reason. For example, an implementation may limit the maximum size of the address space which can be processed during a single RMI command, either to ensure that the command's execution time does not exceed an IMPLEMENTATION DEFINED budget, or to limit complexity of the implementation. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made. If no changes of system state occur before the operation is resumed then the next command will fail the same check at the beginning, and return an error code as described above. This indicates that in order to progress the operation, the Host must first unfold the RTT, transition the tracking region to fine granularity, or create a Level 1 DPT.
- The operation neither encounters a failing check, nor reaches the end of the IPA range, but yields for an IMPLEMENTATION DEFINED reason. For example, this could be due to the command's execution time having exceeded an IMPLEMENTATION DEFINED budget. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made.
- The operation reaches the end of the target IPA range. The command returns RMI_SUCCESS with `out_top == top`.

I_{WYJKK} A Range-based memory mapping operation exclusively targets either Protected IPA space or Unprotected IPA space.

- RMI_RTT_UNPROT_MAP targets Unprotected IPA space.
- All other Range-based memory mapping operations target Protected IPA space.

I_{VRXMQ} On successful execution of a Range-based memory mapping operation, all of the following are true:

- The HIPAS of the IPA range `[base, out_top)` transitions from HIPAS_VOID to the target HIPAS value. For example, when creating a mapping to conventional memory, the target HIPAS value is HIPAS_DATA.
- If the operation uses an output address set then the IPA range `[base, out_top)` is mapped linearly to the first `out_top - base` bytes of the output address set.
- If the operation uses an output address set then the state of the Granules within the first `out_top - base` bytes of the output address set transitions from GRAN_DELEGATED to the target Granule state. For example, when creating a mapping to conventional memory, the target Granule state is GRAN_DATA.

I_{HCNBP} If the return value of a Range-based memory mapping operation is RMI_INCOMPLETE and the operation uses an RMI Address Range List then it is IMPLEMENTATION DEFINED whether data is read from the list.

A5.3.3.3 Range-based memory unmapping operation

I_{BWPHJ} The amount of progress made on a Range-based memory unmapping operation during a single RMI call is determined by the combination of constraints imposed by the RMM architecture, and IMPLEMENTATION DEFINED properties of the RMM implementation.

The following architectural constraints apply to the RTTE at the base of the target IPA range:

- If base of the target IPA range is not aligned to the size of the IPA space described by the RTTE then the command fails with (RMI_ERROR_RTT, walk.level).
- If the size of the IPA space described by the RTTE is larger than the total size of the target IPA range then the command fails with (RMI_ERROR_RTT, walk.level).

If the above checks pass then the start of the output address set is read from the above RTTE.

If the operation is removing DATA or RIPAS_DEV mappings then the following architectural constraints apply to the tracking region which includes the start of the set:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- If start of the output address set is not aligned to the granularity of the tracking region then the command fails with (RMI_ERROR_RTT, walk.level).
- If the granularity of the tracking region is larger than the total size of the target IPA range then the command fails with RMI_ERROR_TRACKING.

If the operation removing DATA or RIPAS_DEV mappings then the following architectural constraints apply to the Granule at the start of the set:

- If the state of the Granule is not the expected Granule state then the command fails with RMI_ERROR_INPUT. For example, when removing a mapping to conventional memory, the expected Granule state is GRAN_DATA.

If the operation removing DATA or RIPAS_DEV mappings then and ATS is enabled for the target Realm then the following architectural constraints apply to the DPT entry which includes the start of the set:

- If the Level 0 DPT is in an intermediate state then the command fails with RMI_BLOCKED.
- If the DPT entry is in an intermediate state then the command fails with RMI_BLOCKED.
- If start of the output address set is not aligned to the size of the PA space described by the DPT entry then the command fails with RMI_ERROR_DPT.
- If the size of the PA space described by the DPT entry is larger than the total size of the target IPA range then the command fails with (RMI_ERROR_RTT, walk.level).

If none of the above checks failed then the operation starts to operate on the target IPA range. The operation terminates when any of the following is true:

- The operation has started to transition an RTTE entry to RTTE_VOID, but has not completed the work needed. The RTTE entry transitions to an intermediate state and the command returns RMI_INCOMPLETE.
- The operation reaches an RTTE, tracking region, Granule or DPT entry which fails one of the above checks. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made. If no changes of system state occur before the operation is resumed then the next command will fail the same check at the beginning, and return an error code as described above.
- The operation reaches an RTTE, tracking region or DPT entry which passes the above checks, but whose size is too large to process for an IMPLEMENTATION DEFINED reason. For example, an implementation may limit the maximum size of the address space which can be processed during a single RMI command, either to ensure that the command's execution time does not exceed an IMPLEMENTATION DEFINED budget, or to limit complexity of the implementation. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made. If no changes of system state occur before the operation is resumed then the next command will fail with RMI_ERROR_RTT, RMI_ERROR_TRACKING or RMI_ERROR_DPT. This indicates that in order to progress the operation, the Host must first unfold the RTT, transition the tracking region to fine granularity, or create a Level 1 DPT.
- The operation neither encounters a failing check, nor reaches the end of the IPA range, but yields for an IMPLEMENTATION DEFINED reason. For example, this could be due to the command's execution time having exceeded an IMPLEMENTATION DEFINED budget. The command returns RMI_SUCCESS with `out_top` indicating the amount of progress made.
- The operation reaches the end of the target IPA range. The command returns RMI_SUCCESS with `out_top == top`.

I_{WQHMZ} A Range-based memory unmapping operation exclusively targets either Protected IPA space or Unprotected IPA space.

- RMI_RTT_UNPROT_UNMAP targets Unprotected IPA space.
- All other Range-based memory mapping operations target Protected IPA space.

I_{WLYBJ} On successful execution of a Range-based memory unmapping operation, all of the following are true:

- The HIPAS of the IPA range `[base, out_top)` transitions to HIPAS_VOID.
- If the Host has requested the output address set then the first `out_top - base` bytes of the output address set are returned, either as the base of a single contiguous PA range, or by populating an RMI Address Range

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

List and returning the number of entries which have been written.

- If the operation uses an output address set then the state of the Granules within the first `out_top - base` bytes of the output address set transitions to GRAN_DELEGATED.

I_{QPWSR} If the return value of a Range-based memory unmapping operation is RMI_INCOMPLETE and the operation uses an RMI Address Range List then it is IMPLEMENTATION DEFINED whether data is written to the list.

A5.3.4 Mapping initial Realm image in Protected IPA space

I_{MLHNJ} RMI_RTT_DATA_MAP_INIT is not a Range-based memory mapping operation.

I_{RGJSN} Execution of RMI_RTT_DATA_MAP_INIT creates a mapping from a page Protected IPA space to conventional memory which is in Realm PAS.

I_{QGGXX} On execution of RMI_RTT_DATA_MAP_INIT, contents of the target conventional memory are initialized with contents provided by the Host.

I_{GNLVS} Execution of RMI_RTT_DATA_MAP_INIT modifies the RIM of the target Realm.

I_{DPWJH} RMI_RTT_DATA_MAP_INIT can be executed only when the state of the target Realm is REALM_NEW.

R_{QDPHR} On execution of RMI_RTT_DATA_MAP_INIT, if the state of the target RTTE is not RTTE_VOID then the command fails with (RMI_ERROR_RTT, walk.level).

R_{FPBKD} On execution of RMI_RTT_DATA_MAP_INIT, if the tracking region for the target output address is not tracked with fine granularity then the command fails with RMI_ERROR_TRACKING.

R_{BXHRK} On execution of RMI_RTT_DATA_MAP_INIT, if the state of the Granule at the target output address is not GRAN_DELEGATED then the command fails with RMI_ERROR_INPUT.

R_{CVGHZ} On execution of RMI_RTT_DATA_MAP_INIT, if ATS is enabled for the target Realm and the DPT entry for the target output address is in an intermediate state then the command fails with RMI_BLOCKED.

R_{TNRZH} On execution of RMI_RTT_DATA_MAP_INIT, if ATS is enabled for the target Realm and the Level 1 DPT has not been created then the command fails with RMI_ERROR_DPT.

R_{TDBWS} On successful execution of RMI_RTT_DATA_MAP_INIT, the target IPA transitions from HIPAS_VOID to HIPAS_DATA.

R_{MPZTK} On successful execution of RMI_RTT_DATA_MAP_INIT, the target IPA is mapped to the target output address.

R_{DGFMY} On successful execution of RMI_RTT_DATA_MAP_INIT, the state of the Granule at the target output address transitions from GRAN_DELEGATED to GRAN_DATA.

I_{GSXTP} RMI_RTT_DATA_MAP_INIT implements the *RMI operations which can lead to an intermediate state* programming model.

See also:

- A5.3.6 *Remove mappings from Protected IPA space to conventional memory*
- B4.3.2 *Stateful RMI operations*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*

A5.3.5 Create mappings from Protected IPA space to wiped conventional memory

I_{DSDWH} Execution of RMI_RTT_DATA_MAP creates mappings from a contiguous range of Protected IPA space to conventional memory which is in Realm PAS.

I_{JZVPQ} RMI_RTT_DATA_MAP is a Range-based memory mapping operation which uses an output address set.

I_{NKXWT} RMI_RTT_DATA_MAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{FPQRZ} Execution of RMI_RTT_DATA_MAP does not modify the RIM of the target Realm.

See also:

- A2.3.8 *Granule wiping*
- A5.3.3 *Range-based memory operations*
- A5.3.6 *Remove mappings from Protected IPA space to conventional memory*
- B4.5.57 *RMI_RTT_DATA_MAP command*

A5.3.6 Remove mappings from Protected IPA space to conventional memory

I_{RDQXH} Execution of RMI_RTT_DATA_UNMAP removes mappings to conventional memory from a contiguous range of Protected IPA space.

I_{LYGYD} RMI_RTT_DATA_UNMAP is a Range-based memory unmapping operation which uses an output address set.

I_{ZNJNQ} RMI_RTT_DATA_UNMAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{RFGPW} Execution of RMI_RTT_DATA_UNMAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.4 *Mapping initial Realm image in Protected IPA space*
- A5.3.5 *Create mappings from Protected IPA space to wiped conventional memory*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*

A5.3.7 Create mappings from Protected IPA space to device memory

I_{HSDRR} Execution of RMI_RTT_DEV_MAP creates mappings from a contiguous range of Protected IPA space to device memory which is in Realm PAS.

I_{DQYFW} RMI_RTT_DEV_MAP is a Range-based memory mapping operation which uses an output address set.

I_{CXPPN} RMI_RTT_DEV_MAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{DVLFV} If RMI_RTT_DEV_MAP reaches an output address which is not within the address ranges of the target VDEV then the command fails with RMI_ERROR_INPUT.

I_{JMBJL} Execution of RMI_RTT_DEV_MAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.8 *Remove mappings from Protected IPA space to device memory*
- Chapter A9 *Realm device assignment*
- B4.5.61 *RMI_RTT_DEV_MAP command*

A5.3.8 Remove mappings from Protected IPA space to device memory

I_{PQJZG} Execution of RMI_RTT_DEV_UNMAP removes mappings to device memory from a contiguous range of Protected IPA space.

I_{VWLSQ} RMI_RTT_DEV_UNMAP is a Range-based memory unmapping operation which uses an output address set.

I_{CDBSP} RMI_RTT_DEV_UNMAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{WSNLS} If RMI_RTT_DEV_UNMAP reaches an output address which is not within the address ranges of the target VDEV then the command fails with RMI_ERROR_INPUT.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{CKCGK} Execution of RMI_RTT_DEV_UNMAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.7 *Create mappings from Protected IPA space to device memory*
- Chapter A9 *Realm device assignment*
- B4.5.62 *RMI_RTT_DEV_UNMAP command*

A5.3.9 Create mappings from Protected IPA space to an architectural device

I_{KSWRM} Execution of RMI_RTT_ARCH_DEV_MAP creates mappings from a contiguous range of Protected IPA space to an architectural device.

I_{WNLDF} In this version of the specification, the only supported type of architectural device is SMMUv3.

I_{NLHMS} RMI_RTT_ARCH_DEV_MAP is a Range-based memory mapping operation which does not use an output address set.

I_{QCYDD} RMI_RTT_ARCH_DEV_MAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{WLRSK} Execution of RMI_RTT_ARCH_DEV_MAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.10 *Remove mappings from Protected IPA space to an architectural device*
- A9.8 *Virtual SMMU*
- B4.5.47 *RMI_RTT_ARCH_DEV_MAP command*

A5.3.10 Remove mappings from Protected IPA space to an architectural device

I_{VZXRD} Execution of RMI_RTT_ARCH_DEV_UNMAP removes mappings to an architectural device from a contiguous range of Protected IPA space.

I_{TMVLH} RMI_RTT_ARCH_DEV_UNMAP is a Range-based memory unmapping operation which does not use an output address set.

I_{VQWHL} RMI_RTT_ARCH_DEV_UNMAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{ZBTRN} Execution of RMI_RTT_ARCH_DEV_UNMAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.9 *Create mappings from Protected IPA space to an architectural device*
- A9.8 *Virtual SMMU*
- B4.5.48 *RMI_RTT_ARCH_DEV_UNMAP command*

A5.3.11 Create mappings from Unprotected IPA space

I_{GZKZN} Execution of RMI_RTT_UNPROT_MAP creates mappings from a contiguous range of Unprotected IPA space to conventional memory which is in Non-Secure PAS.

I_{QLBDR} RMI_RTT_UNPROT_MAP is a Range-based memory mapping operation which uses an output address set.

I_{ZRHBP} RMI_RTT_UNPROT_MAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{TVTMR} Execution of RMI_RTT_UNPROT_MAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.12 *Remove mappings from Unprotected IPA space*
- B4.5.69 *RMI_RTT_UNPROT_MAP command*

A5.3.12 Remove mappings from Unprotected IPA space

I_{VRXCD} Execution of RMI_RTT_UNPROT_UNMAP removes mappings to conventional memory from a contiguous range of Unprotected IPA space.

I_{RRJNY} RMI_RTT_UNPROT_UNMAP is a Range-based memory unmapping operation which uses an output address set.

I_{QMMTB} RMI_RTT_UNPROT_UNMAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{CLVCM} Execution of RMI_RTT_UNPROT_UNMAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.11 *Create mappings from Unprotected IPA space*
- B4.5.70 *RMI_RTT_UNPROT_UNMAP command*

A5.3.13 Create mappings within Protected IPA space in auxiliary RTT tree

I_{BQRQH} Execution of RMI_RTT_AUX_PROT_MAP propagates mappings within a contiguous range of Protected IPA space from the primary RTT tree to an auxiliary RTT tree.

I_{CKYCM} RMI_RTT_AUX_PROT_MAP is a Range-based memory mapping operation which does not use an output address set.

I_{XGMXX} RMI_RTT_AUX_PROT_MAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{LPYYG} On execution of RMI_RTT_AUX_PROT_MAP, the “block” flag controls the behaviour when the auxiliary RTT walk reaches an entry which extends beyond the start or the end of the target IPA range.

- If the value is RMI_RTT_AUX_BLOCK_CREATE then the RMM modifies the auxiliary RTT entry. The result is that the IPA range which is mapped extends beyond the start or the end of the target IPA range.
- If the value is RMI_RTT_AUX_BLOCK_NO_CREATE then the command fails and returns an error to the caller. The expected result is that the Host will unfold the auxiliary RTT tree before re-trying the mapping operation.

I_{LVBNM} On execution of RMI_RTT_AUX_PROT_MAP, the “invalid_pri” flag controls the behaviour when the primary RTT walk reaches an entry whose state is RTTE_VOID.

- If the value is RMI_RTT_AUX_INVALID_CONTINUE then the RMM skips the corresponding entry in the auxiliary RTT and continues the mapping operation.
- If the value is RMI_RTT_AUX_INVALID_STOP then the command fails and returns an error to the caller.

I_{XTPQV} Execution of RMI_RTT_AUX_PROT_MAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.14 *Remove mappings within Protected IPA space in auxiliary RTT tree*
- B4.5.52 *RMI_RTT_AUX_PROT_MAP command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A5.3.14 Remove mappings within Protected IPA space in auxiliary RTT tree

I_{PTPPN} Execution of RMI_RTT_AUX_PROT_UNMAP removes mappings within a contiguous range of Protected IPA space from an auxiliary RTT tree.

I_{BFRKG} RMI_RTT_AUX_PROT_UNMAP is a Range-based memory mapping operation which does not use an output address set.

I_{NZHMK} RMI_RTT_AUX_PROT_UNMAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{QWNXW} Execution of RMI_RTT_AUX_PROT_UNMAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.13 *Create mappings within Protected IPA space in auxiliary RTT tree*
- B4.5.53 *RMI_RTT_AUX_PROT_UNMAP command*

A5.3.15 Create mappings within Unprotected IPA space in auxiliary RTT tree

I_{DTGQS} Execution of RMI_RTT_AUX_UNPROT_MAP propagates mappings within a contiguous range of Unprotected IPA space from the primary RTT tree to an auxiliary RTT tree.

I_{KXHFT} RMI_RTT_AUX_UNPROT_MAP is a Range-based memory mapping operation which does not use an output address set.

I_{VXQXC} RMI_RTT_AUX_UNPROT_MAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{BLXRL} Execution of RMI_RTT_AUX_UNPROT_MAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.16 *Remove mappings within Unprotected IPA space in auxiliary RTT tree*
- B4.5.54 *RMI_RTT_AUX_UNPROT_MAP command*

DRAFT

A5.3.16 Remove mappings within Unprotected IPA space in auxiliary RTT tree

I_{KMGVK} Execution of RMI_RTT_AUX_UNPROT_UNMAP removes mappings within a contiguous range of Unprotected IPA space from an auxiliary RTT tree.

I_{HSHRL} RMI_RTT_AUX_UNPROT_UNMAP is a Range-based memory mapping operation which does not use an output address set.

I_{XVKCY} RMI_RTT_AUX_UNPROT_UNMAP can be executed either when the state of the target Realm is REALM_NEW or when the state of the target Realm is REALM_ACTIVE.

I_{WXJQM} Execution of RMI_RTT_AUX_UNPROT_UNMAP does not modify the RIM of the target Realm.

See also:

- A5.3.3 *Range-based memory operations*
- A5.3.15 *Create mappings within Unprotected IPA space in auxiliary RTT tree*
- B4.5.55 *RMI_RTT_AUX_UNPROT_UNMAP command*

A5.3.17 Changes to HIPAS of a Protected IPA

A5.3.17.1 Changes to HIPAS of a Protected IPA 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

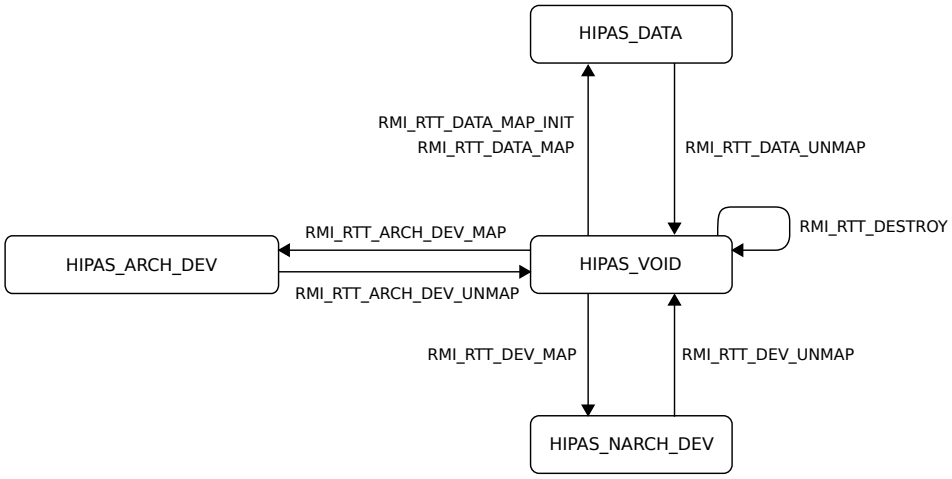


Figure A5.2: HIPAS changes at a Protected IPA which can occur when the Realm state is REALM_NEW

See also:

- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.61 *RMI_RTT_DEV_MAP command*
- B4.5.62 *RMI_RTT_DEV_UNMAP command*

DRAFT

A5.3.17.2 Changes to HIPAS of a Protected IPA 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.

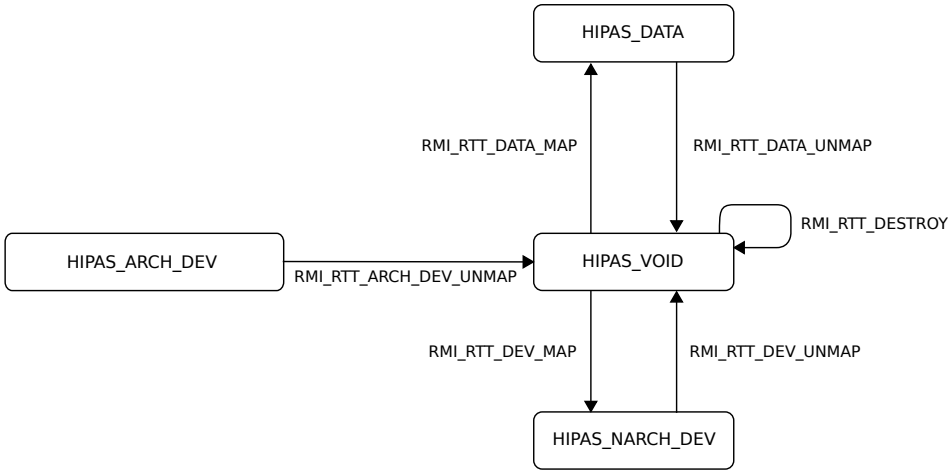


Figure A5.3: HIPAS changes at a Protected IPA which can occur when the Realm state is REALM_ACTIVE

See also:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.61 *RMI_RTT_DEV_MAP command*
- B4.5.62 *RMI_RTT_DEV_UNMAP command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A5.3.18 Summary of changes to HIPAS and RIPAS of a Protected IPA

I_{TJMCP} The following diagram summarizes HIPAS and RIPAS changes at a Protected IPA which can occur when the Realm state is REALM_NEW.

Transitions due to execution of RMI_RTT_DESTROY are omitted from the diagram. Execution of this command results in a transition to HIPAS_VOID, RIPAS_DESTROYED.

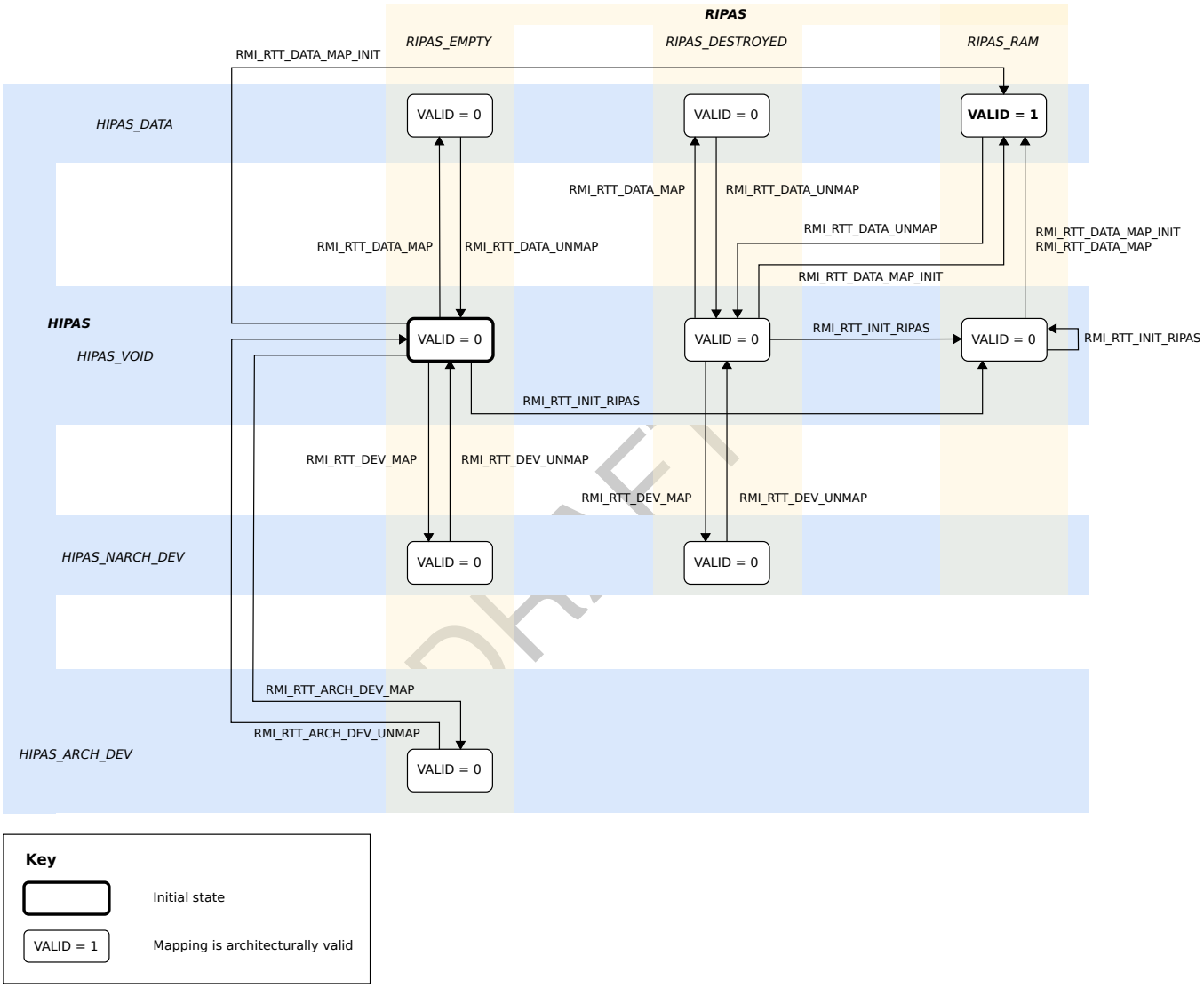


Figure A5.4: HIPAS and RIPAS changes at a Protected IPA which can occur when the Realm state is REALM_NEW

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

IVGKNJ 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_VOID, RIPAS_DESTROYED.

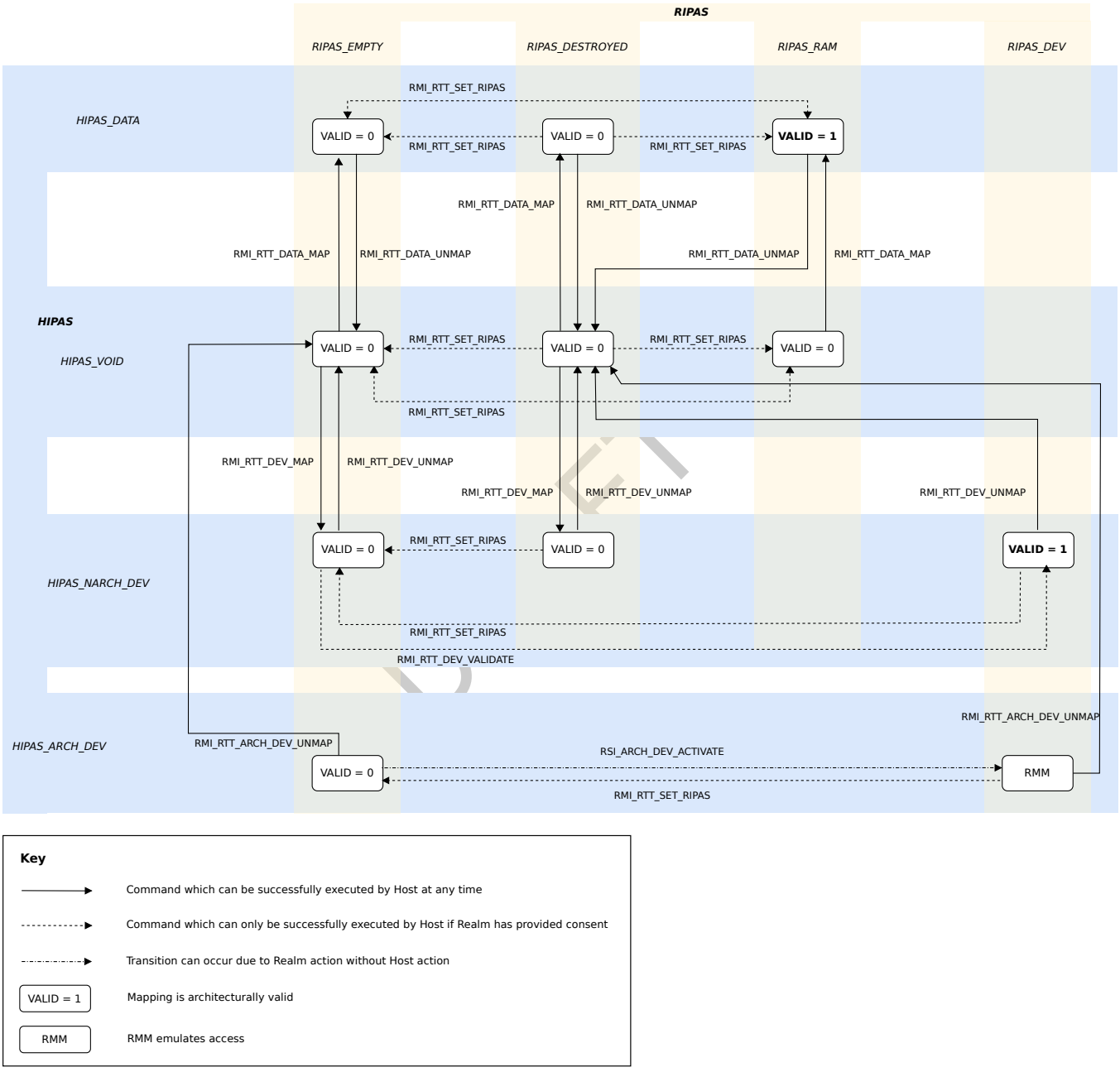


Figure A5.5: HIPAS and RIPAS changes at a Protected IPA which can occur when the Realm state is REALM_ACTIVE

See also:

- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.61 *RMI_RTT_DEV_MAP command*
- B4.5.62 *RMI_RTT_DEV_UNMAP command*
- B4.5.63 *RMI_RTT_DEV_VALIDATE command*
- B4.5.65 *RMI_RTT_INIT_RIPAS command*
- B4.5.67 *RMI_RTT_SET_RIPAS command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A5.3.19 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_RTT_ARCH_DEV_MAP	EMPTY	VOID	Unchanged	ARCH_DEV
RMI_RTT_ARCH_DEV_UNMAP	Not DEV	ARCH_DEV	Unchanged	VOID
RMI_RTT_ARCH_DEV_UNMAP	DEV	ARCH_DEV	DESTROYED	VOID
RMI_RTT_CREATE	None	None	Unchanged	Unchanged
RMI_RTT_DESTROY	None	HIPAS of all entries is VOID	DESTROYED	VOID
RMI_RTT_DATA_MAP_INIT	None	VOID	RAM	DATA
RMI_RTT_DATA_MAP	None	VOID	Unchanged	DATA
RMI_RTT_DATA_UNMAP	Not RAM	DATA	Unchanged	VOID
RMI_RTT_DATA_UNMAP	RAM	DATA	DESTROYED	VOID
RMI_RTT_DEV_MAP	None	VOID	Unchanged	NARCH_DEV
RMI_RTT_DEV_UNMAP	Not DEV	NARCH_DEV	Unchanged	VOID
RMI_RTT_DEV_UNMAP	DEV	NARCH_DEV	DESTROYED	VOID
RMI_RTT_FOLD	RIPAS of all entries is identical	HIPAS of all entries is identical	Unchanged	Unchanged
RMI_RTT_INIT_RIPAS	None	VOID	RAM	Unchanged
RMI_RTT_SET_RIPAS	Optionally, Realm may specify that RIPAS is not DESTROYED	None	As specified by Realm	Unchanged
RMI_RTT_DEV_VALIDATE	None	HIPAS of all entries is NARCH_DEV	DEV	Unchanged

DRAFT

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

I_{LCSVH} Successful execution of RMI_RTT_DATA_UNMAP 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.

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

See also:

- B4.5.47 *RMI_RTT_ARCH_DEV_MAP command*
- B4.5.48 *RMI_RTT_ARCH_DEV_UNMAP command*
- B4.5.56 *RMI_RTT_CREATE command*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.61 *RMI_RTT_DEV_MAP command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.62 *RMI_RTT_DEV_UNMAP command*
- B4.5.63 *RMI_RTT_DEV_VALIDATE command*
- B4.5.64 *RMI_RTT_FOLD command*
- B4.5.65 *RMI_RTT_INIT_RIPAS command*
- B4.5.67 *RMI_RTT_SET_RIPAS command*

A5.3.20 Changes to HIPAS of an Unprotected IPA

IYNYBY The following diagram summarises HIPAS transitions for an Unprotected IPA.

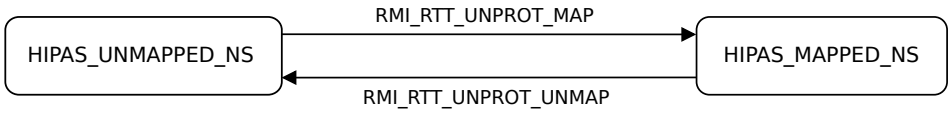


Figure A5.6: HIPAS transitions for an Unprotected IPA

See also:

- A5.6 *Realm Translation Table*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.65 *RMI_RTT_INIT_RIPAS command*
- B4.5.67 *RMI_RTT_SET_RIPAS command*
- B5.4.7 *RSI_IPA_STATE_SET command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 an RSI command.
  - The input values to the RSI command include the requested IPA range: `[base, top)`.
  - The target RIPAS value is either passed as an input, or is implied by the command.
  - The RMM records the request the REC, and then performs a REC exit.
- In response, the Host executes zero or more RMI commands to apply the RIPAS change.
- If the requested RIPAS value was not RIPAS_EMPTY then at the next RMI_REC_ENTER the Host can optionally indicate that it rejects the RIPAS change request.

Output values from the RSI command indicate:

- The top of the IPA range which has been modified by the command (`new_base`).
- Whether the Host accepted or rejected the Realm request.

See also:

- A5.2.2 *Realm IPA state*

A5.4.1 Realm view of RIPAS change

I_{CKBLH} The RSI commands which can initiate a RIPAS change request are:

- RSI_IPA_STATE_SET
  - The target RIPAS value, either RIPAS_EMPTY or RIPAS_RAM, is provided as an input value.
  - If the target RIPAS value is RIPAS_RAM, a flag indicates whether a change from RIPAS_DESTROYED should be permitted.
- RSI_VDEV_VALIDATE_MAPPING
  - The target RIPAS value is RIPAS_DEV.

I_{HXKPB} On REC entry following a REC exit due to a RIPAS change request, GPR values are updated to indicate for how much of the target IPA range the RIPAS change has been applied.

S_{TZYZV} To complete a RIPAS change for a given target IPA range, a Realm should execute the initiating command in a loop, until the value of X1 reaches the top of the target IPA range.

S_{BZWWC} Receipt of a rejection for a RIPAS change request whose parameters were valid is expected to be fatal for the Realm.

S_{CTTQV} Output values from the initiating RSI command are expected to be handled by the Realm as follows:

new_base	**response**	**Meaning**	**Expected Realm action**
`new_base == base`	RSI_RESPONSE_ACCEPT	RIPAS change incomplete.	Call the command again, with `base = new_base`.
`base < new_base < top`	RSI_RESPONSE_ACCEPT	RIPAS change incomplete.	Call the command again, with `base = new_base`.
`new_base == top`	RSI_RESPONSE_ACCEPT	RIPAS change complete.	No further Realm action required.
`new_base == base`	RSI_RESPONSE_REJECT	RIPAS change request rejected.	Depends on protocol agreed between Realm and Host, out of scope of this specification.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

new_base	response	Meaning	Expected Realm action
base < new_base < top	RSI_RESPONSE_REJECT	RIPAS change to partial region [base, new_base). Host rejected request to change RIPAS for region [new_base, top).	Depends on protocol agreed between Realm and Host, out of scope of this specification.

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

See also:

- A7.1.1 *Realm Initial Measurement*
- B5.4.7 *RSI_IPA_STATE_SET command*
- B5.4.20 *RSI_VDEV_VALIDATE_MAPPING command*

A5.4.2 Host view of RIPAS change to RIPAS_EMPTY or RIPAS_RAM

I_{WBCJL} A RIPAS change request whose target is RIPAS_EMPTY or RIPAS_RAM results in a REC exit due to RIPAS change.

I_{LPZWK} A RIPAS change whose target is RIPAS_EMPTY or RIPAS_RAM 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

I_{GXDDX} If all of the following are true on successful execution of RMI_RTT_SET_RIPAS

- The target RIPAS is RIPAS_RAM
- The RIPAS change request indicated that a change from RIPAS_DESTROYED to RIPAS_RAM should not be permitted
- A page *P* within the target IPA range has RIPAS value RIPAS_DESTROYED

then rec.ripas_addr and the command output value are both set to *P*.

R_{LDMLC} On REC entry following a REC exit due to RIPAS change, rec.ripas_response is set to the value of enter.flags.ripas_response.

I_{DRPPK} If all of the following are true then the output value of RSI_IPA_STATE_SET indicates “Host rejected the request”:

- rec.ripas_value is RIPAS_RAM.
- rec.ripas_addr is not equal to rec.ripas_top.
- rec.ripas_response is REJECT.

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

See also:

- A4.3.8 *REC exit due to RIPAS change pending*
- B4.5.42 *RMI_REC_ENTER command*
- B4.5.67 *RMI_RTT_SET_RIPAS command*
- D1.5.3 *RIPAS change flow*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A5.4.3 Host view of RIPAS change to RIPAS_DEV

I_{TLZWR} A RIPAS change request whose target is RIPAS_DEV results in a REC exit due to VDEV mapping validation.

I_{BJBJB} A RIPAS change whose target is RIPAS_DEV is applied by one or more calls to the RMI_RTT_DEV_VALIDATE command.

See also:

- A4.3.13 *REC exit due to VDEV mapping validation*
- A5.5 *VDEV mapping validation*
- A9.6.2 *Realm validation of device memory mappings*
- B4.5.42 *RMI_REC_ENTER command*
- B4.5.63 *RMI_RTT_DEV_VALIDATE command*
- D1.5.3 *RIPAS change flow*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A5.5 VDEV mapping validation

D_{NZSKS} A *VDEV 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.

I_{FZHYH} The outcome of a successful VDEV mapping validation is a change of RIPAS to RIPAS_DEV.

I_{LLPZR} A VDEV mapping validation consists of actions taken first by the Realm, and then by the Host:

- The Realm issues a *VDEV mapping validation request* by executing RSI_VDEV_VALIDATE_MAPPING.
  - The input values to RSI_VDEV_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 VDEV mapping validation.
- In response, the Host executes zero or more RMI_RTT_DEV_VALIDATE commands.

Output values from RSI_VDEV_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.

S_{RCNJD} Output values from RSI_VDEV_VALIDATE_MAPPING are expected to be handled by the Realm as follows:

new_base	**response**	**Meaning**	**Expected Realm action**
`new_base == base`	RSI_RESPONSE_ACCEPT	VDEV mapping validation incomplete.	Call the command again, with `base = new_base`.
`base < new_base < top`	RSI_RESPONSE_ACCEPT	VDEV mapping validation incomplete.	Call the command again, with `base = new_base`.
`new_base == top`	RSI_RESPONSE_ACCEPT	VDEV mapping validation complete.	No further Realm action required.
`new_base == base`	RSI_RESPONSE_REJECT	VDEV mapping validation request rejected.	Depends on protocol agreed between Realm and Host, out of scope of this specification.
`base < new_base < top`	RSI_RESPONSE_REJECT	VDEV 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.

DRAFT

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

I_{YBJZF} A VDEV mapping validation is applied by one or more calls to the RMI_RTT_DEV_VALIDATE command.

I_{TLRPR} Successful execution of RMI_RTT_DEV_VALIDATE targets an RTTE at address `rec.dev_mem_addr`.

I_{YVGNP} On successful execution of RMI_RTT_DEV_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`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- `rec.dev_mem_pa`
- The command output value

I_{SGKTK} On REC entry following a REC exit due to VDEV mapping validation, GPR values are updated to indicate for how much of the target IPA range the VDEV mapping validation request has been applied.

S_{MWWXS} To complete a VDEV mapping validation for a given target IPA range, a Realm should execute RSI_VDEV_VALIDATE_MAPPING in a loop, until the value of X1 reaches the top of the target IPA range.

R_{FDCBZ} On REC entry following a REC exit due to VDEV mapping validation, `rec.dev_mem_response` is set to the value of
`enter.flags.dev_mem_response`.

I_{LDDPH} If all of the following are true then the output value of RSI_VDEV_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_VDEV_VALIDATE_MAPPING indicates “Host accepted the request”.

See also:

- A2.4.2 *REC attributes*
- A4.2 *REC entry*
- A4.3.13 *REC exit due to VDEV mapping validation*
- A5.2.2 *Realm IPA state*
- A7.1.1 *Realm Initial Measurement*
- Chapter A9 *Realm device assignment*
- B3.100 *RecDevMemResponseToRsi function*
- B4.5.42 *RMI_REC_ENTER command*
- B4.5.63 *RMI_RTT_DEV_VALIDATE command*
- B5.4.20 *RSI_VDEV_VALIDATE_MAPPING command*
- D1.5.3 *RIPAS change flow*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 *Realm Translation Table* (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 equal to the RMI Granule size.

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 *RTT entry* (RTTE) is an abstraction over an Armv8-A stage 2 translation table descriptor.

I_{VYLTT} An RTTE contains an output address which can point to one of the following:

- 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 *RTT tree* is a hierarchical data structure composed of RTTs, connected via Table Descriptors.

I_{KNPNX} An RTT contains an array of RTTEs.

D_{HYTCJ} An *RTT level* 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 *starting level*.

I_{SSDBT} The maximum depth of an RTT tree depends on all of the following:

- whether LPA2 is selected when the Realm is created
- the rtt_level_start attribute of the Realm
- the ipa_width attribute of the Realm.

I_{QRVTT} The VMID associated with an RTT tree is chosen by the RMM. The RMM ensures that this is unique among all RTT trees on the system.

See also:

- A2.2.3 *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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

See also:

- A2.2.3 *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
RTTE_ARCH_DEV	This RTTE is identified by a Protected IPA. The output address of this RTTE points to an RMM object which is used to emulate an architectural device.
RTTE_AUX_DESTROYED	An auxiliary RTT was destroyed while a corresponding primary RTT entry was live.
RTTE_DATA	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a DATA Granule.
RTTE_MAPPED_NS	This RTTE is identified by an Unprotected IPA. The output address of this RTTE points to a Granule-aligned address within NS PAS.
RTTE_NARCH_DEV	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a GRAN_DEV Granule.
RTTE_TABLE	The output address of this RTTE points to the next-level RTT.
RTTE_UNMAPPED_NS	This RTTE is identified by an Unprotected IPA. This RTTE is not associated with any Granule.
RTTE_VOID	This RTTE is identified by a Protected 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 RTTE_VOID, RTTE_UNMAPPED_NS or RTTE_TABLE.

D_{NSHSL} The output address of an RTTE whose state is RTTE_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 RTTE_TABLE RTTE in a level *n-1* RTT.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{DXQWZ} The following diagram shows an example RTT tree, annotated with RTTE states.

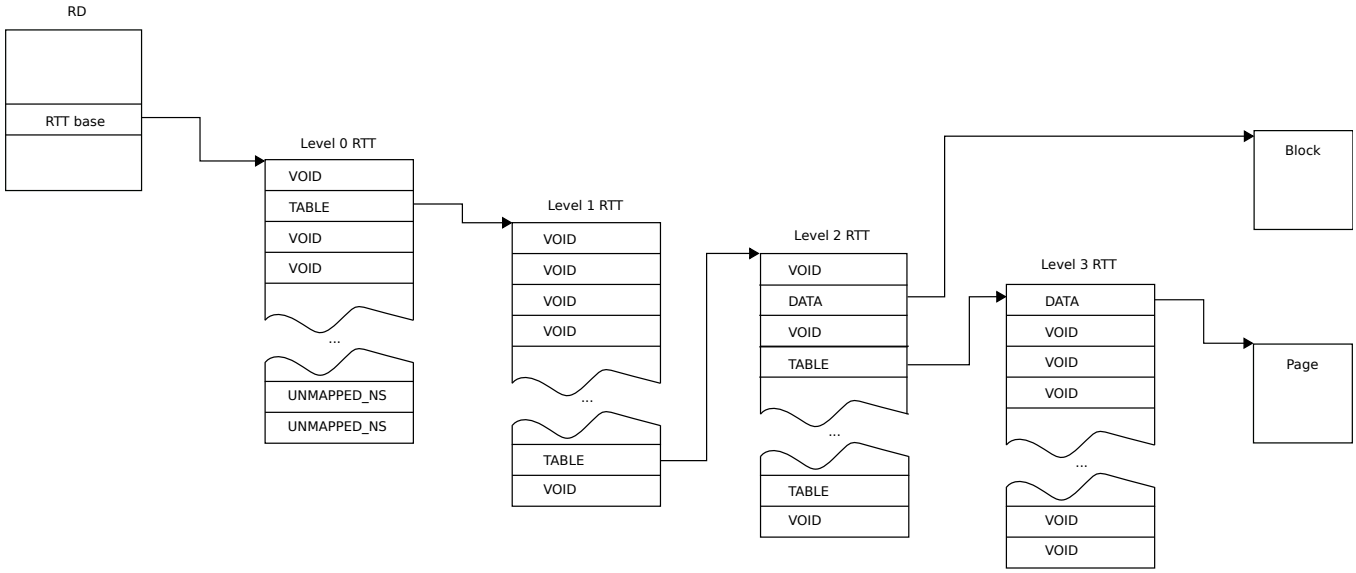


Figure A5.7: Example RTT tree

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*

DRAFT

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:

- B4.5.66 *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 RTTE_VOID or RTTE_UNMAPPED_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_{QPXCP} *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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{QMGWK} On RTT folding, the state of the parent RTTE is set to the state of the child RTTEs.

I_{YRJTT} On RTT folding, if the RTT describes Protected IPA space then the RIPAS of the parent RTTE is set to the RIPAS of the child RTTEs.

I_{QXKYY} On RTT folding, the S2AP fields of the parent RTTE is set to the S2AP fields of the child RTTEs.

I_{TPMGT} On RTT folding, if the state of the parent RTTE is not RTTE_VOID or RTTE_UNMAPPED_NS then all of the following are true:

- The output address of the parent RTTE is set to the output address of the first child RTTE.
- The memory attributes of the parent RTTE are set to the memory attributes of the child RTTEs.

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

See also:

- A5.6.9 *RTT destruction*
- A10.3.2 *Stage 2 Access Permissions within a multi-Plane Realm*
- B3.173 *RttFold function*
- B3.174 *RttIsHomogeneous function*
- B4.5.64 *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_{SPVFG} On RTT unfolding, if the RTT describes Protected IPA space then the RIPAS of all RTTEs in the child RTT are set to the state of the parent RTTE.

I_{BLLVF} On RTT unfolding, the S2AP fields of all RTTEs in the child RTT are set to the S2AP fields of the parent RTTE.

I_{HMYSW} On RTT unfolding, if the state of the parent RTTE is not RTTE_VOID or RTTE_UNMAPPED_NS then all of the following are true:

- The output addresses of RTTEs in the child RTT are set to a contiguous range which starts from the address of the parent RTTE.
- The memory attributes of all RTTEs in the child RTT are set to the memory attributes of the parent RTTE.

See also:

- B4.5.56 *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:

- RTTE_DATA
- RTTE_MAPPED_NS
- RTTE_NARCH_DEV
- RTTE_TABLE
- RTTE_ARCH_DEV

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- RTTE_DATA
- RTTE_NARCH_DEV
- RTTE_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 RTTE_MAPPED_NS.

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

See also:

- A5.6.9 *RTT destruction*
- B3.175 *RttIsLive function*
- B3.190 *RttSkipNonLiveEntries function*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.70 *RMI_RTT_UNPROT_UNMAP 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 within Protected IPA space, all of the following are true for the parent RTTE:

- RIPAS is RIPAS_DESTROYED
- RTTE state is RTTE_VOID

I_{RTGXB} Following RTT destruction within Unprotected IPA space, the state of the parent RTTE is RTTE_UNMAPPED_NS.

See also:

- A5.2 *Realm view of memory management*
- A5.6.6 *RTT folding*
- A5.6.8 *RTTE liveness and RTT liveness*
- B4.5.60 *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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- it reaches an RTTE whose state is not RTTE_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:

- A2.2.3 *Realm attributes*
- A10.3.1 *Auxiliary RTT*
- B1.4 *Command condition expressions*
- B3.202 *RttWalk function*
- B4.5.56 *RMI_RTT_CREATE command*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.69 *RMI_RTT_UNPROT_MAP command*
- B4.5.70 *RMI_RTT_UNPROT_UNMAP command*
- C2.72 *RmmRttWalkResult type*

DRAFT

A5.6.11 Stage 2 Access Permissions

This section describes how Stage 2 Access Permissions (S2AP) observed by Auxiliary Planes 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.

See also:

- Chapter A10 *Planes*

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

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

I_{SZFTD} 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*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A5.6.11.2 Stage 2 Access Permissions for a Protected IPA

I_{MNGYV} 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].

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

U_{NRGZW} 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.

R_{GNLRT} For a Protected IPA which is HIPAS_DATA, the S2AP base value is *RW+puX*.

R_{FTCJG} For a Protected IPA which is HIPAS_NARCH_DEV, the S2AP base value is *RW*.

R_{NCYRD} 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

I_{XQMYN} The S2AP which applies to a Realm access to an Unprotected IPA is controlled by the Host.

I_{KJWXT} 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_UNPROT_MAP to create the mapping in the primary RTT tree.
- The RMM sets `XN = '1'` in the primary RTT tree.
- Execution of RMI_RTT_AUX_UNPROT_MAP 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_UNPROT_MAP.
- 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*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.54 *RMI_RTT_AUX_UNPROT_MAP command*
- B4.5.69 *RMI_RTT_UNPROT_MAP command*

A5.6.11.4 Stage 2 base permission values

R_{YTRNK} The mapping from S2AP base index to S2AP base value is as follows:

Encoding	Name	Description
0	RMI_S2AP_NO_ACCESS	NoAccess
1	RMI_S2AP_RO	RO
2	RMI_S2AP_WO	WO
3	RMI_S2AP_RW	RW
4	RMI_S2AP_RW_PUX	RW+puX

See also:

- *Arm Architecture Reference Manual for A-Profile architecture* [3]

A5.6.12 Memory attributes

A5.6.12.1 Memory attributes for RTTE_DATA mappings

R_{LBHZG} The memory type and cacheability attributes which result from Realm access to an IPA which is HIPAS_DATA are Normal Write-Back.

R_{KYTFD} The shareability attributes which result from Realm access to an IPA which is HIPAS_DATA are Inner Shareable.

R_{KCFCT} The memory attributes which result from Realm access to an IPA which is HIPAS_DATA 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 which is HIPAS_DATA are independent of stage 1 translation.

See also:

- *Arm Architecture Reference Manual for A-Profile architecture* [3]
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*

A5.6.12.2 Memory attributes for RTTE_NARCH_DEV mappings

R_{LFSWQ} The memory type and cacheability attributes which result from Realm access to an IPA which is HIPAS_NARCH_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:

- Device, with device attributes specified via stage 1 translation
- Normal Non-Cacheable

U_{PTZDY} The RMM uses FEAT_S2FWB to constrain the memory type and cacheability attributes which result from Realm access to an IPA which is HIPAS_NARCH_DEV and which is mapped to a Device non-coherent memory physical location.

R_{KTKVK} The memory type and cacheability attributes which result from Realm access to an IPA which is HIPAS_NARCH_DEV and which maps to a Device coherent memory physical location are passed through from stage 1 translation.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

U_{ZYWFN} 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 which is HIPAS_NARCH_DEV and which is mapped to a Device coherent memory physical location.

R_{KMHLP} The shareability attributes which result from Realm access to an IPA which is HIPAS_NARCH_DEV and which maps to a Device non-coherent memory physical location are Outer Shareable.

R_{JXJTP} The shareability attributes which result from Realm access to an IPA which is HIPAS_NARCH_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]
- A2.3 *Physical memory*
- A9.6.2 *Realm validation of device memory mappings*
- B4.5.61 *RMI_RTT_DEV_MAP command*

A5.6.12.3 Memory attributes for RTTE_MAPPED_NS mappings

D_{FJTMF} The following attributes of an RTT entry whose state is RTTE_MAPPED_NS are *Host-controlled Unprotected RTT attributes*:

- `ADDR`
- `MemAttr[2:0]`
- `AP` (if the Realm uses S2AP direct encoding) or `PIIndex` (if the Realm uses S2AP indirect encoding)

X_{QHLKB} In an RTT entry whose state is RTTE_MAPPED_NS, `MemAttr[3]` is `RES0` because the RMM uses FEAT_S2FWB.

R_{QFLWD} The shareability attributes which result from Realm access to an IPA which is HIPAS_MAPPED_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*
- B3.164 *RttDescriptorIsValidForUnprotected function*
- B4.5.69 *RMI_RTT_UNPROT_MAP command*

A5.6.12.4 Summary of memory attributes

I_{ZNCSK} 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
HIPAS_DATA	Delegable conventional memory	Normal Write-Back	Inner Shareable

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

HIPAS	Granule category	Resultant memory type and cacheability attributes	Resultant shareability attributes
HIPAS_NARCH_DEV	Delegable non-coherent device memory	One of the following: • Device • Normal Non-cacheable (Specified by stage 1 translation)	Outer Shareable
HIPAS_NARCH_DEV	Delegable coherent device memory	Specified by stage 1 translation	Inner Shareable
HIPAS_MAPPED_NS	Delegable conventional memory	Specified by combination of Host-controlled RTT attributes and stage 1 translation	• Inner Shareable if the mapping is cacheable • Outer Shareable if the mapping is non-cacheable

See also:

- A2.3.5 *Delegable physical memory*

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A6
Realm interrupts and timers

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A6.1 Realm interrupts

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

R_{NMPBP} On REC entry, the values of the following registers are preserved:

- `ICH_AP0R<n>_EL2`
- `ICH_AP1R<n>_EL2`
- `ICH_LR<n>_EL2`
- `ICH_VMCR_EL2`
- `ICH_HCR_EL2`

R_{HLFRY} REC entry fails if `ICH_LR<n>_EL2.HW == '1'` for any implemented value of `n`.

X_{DMSDZ} The GICv3 architecture states that, if `HW == '1'` 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 `HW == '0'`.

R_{LNQRL} A REC exit due to IRQ is not generated for an interrupt which is masked by the value of `ICC_PMR_EL1` at the time of REC entry.

R_{FGQXT} On REC exit, the values of the following registers are preserved:

- `ICH_AP0R<n>_EL2`
- `ICH_AP1R<n>_EL2`
- `ICH_LR<n>_EL2`
- `ICH_VMCR_EL2`
- `ICH_HCR_EL2`

R_{VSBBS} On REC exit, `ICH_HCR_EL2.En == '0'`.

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.

R_{XRKLG} Realm write access to any of the following registers results in a REC exit due to System register access:

- `ICC_ASGI1R_EL1`
- `ICC_SGI0R_EL1`
- `ICC_SGI1R_EL1`

R_{MMTQQ} Realm access to `ICC_*_EL1` except for the following registers results in a REC exit due to System register access if the corresponding trap was set in `ICH_HCR_EL2` at the time of the REC entry:

- `ICC_ASGI1R_EL1`
- `ICC_SGI0R_EL1`
- `ICC_SGI1R_EL1`

See also:

- *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*
- B4.5.5 *RMI_FEATURES command*
- B4.5.42 *RMI_REC_ENTER command*
- B4.6.64 *RmiRecEnter type*
- B4.6.66 *RmiRecExit type*
- D1.6.1 *Interrupt flow*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

R_{LKNDV} Architectural timers are available to a Realm and behave according to their architectural specification.

I_{VFYJV} If the Host has programmed an EL1 timer to assert its output during Realm execution, that timer output is not guaranteed to assert.

R_{FKCHX} If the Host has programmed an EL2 timer to assert its output during Realm execution, that timer output is guaranteed to assert.

R_{RJZRP} 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.

R_{JSMQP} 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.

X_{YCDMW} 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.

I_{FKMGZ} 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.

R_{SVCMR} 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.

R_{VWQDH} On REC exit, Realm EL1 timer state is exposed via the RmiRecExit object:

- `exit.cntv_ctl` contains the value of `CNTV_CTL_EL0` at the time of the Realm exit.
- `exit.cntv_cval` contains the value of `CNTV_CVAL_EL0` at the time of the Realm exit, expressed as if the virtual counter offset was zero.
- `exit.cntp_ctl` contains the value of `CNTP_CTL_EL0` at the time of the Realm exit.
- `exit.cntp_cval` contains the value of `CNTP_CVAL_EL0` at the time of the Realm exit, expressed as if the physical counter offset was zero.

S_{PYWWF} The Host should check the Realm EL1 timer state on every return from RMI_REC_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 REC exit due to Data Abort), the Realm EL1 timer state should be checked.

I_{VRWGS} 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- A4.3 *REC exit*
- A10.5 *Planes timers*
- B4.6.66 *RmiRecExit type*
- D1.6.2 *Timer interrupt delivery flow*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A7

Realm measurement and attestation

DRAFT

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.1 Realm measurements

This section describes how Realm measurement values are calculated.

D_{SJWWS} A Realm measurement value is an incremental hash.

D_{YKDBY} A *Realm Hash Algorithm* (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 `hash_algo` attribute.

See also:

- A2.2.3 *Realm attributes*
- A3.2 *Realm hash algorithm*
- A7.2.3.1.5 *Realm Initial Measurement claim*
- A7.2.3.1.6 *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 zero.

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:

- Creation of a DATA Granule during Realm construction
- Creation of a runnable REC

R_{VMPZG} On execution of an operation which requires extension of a RIM, the RMM first constructs a *measurement descriptor* structure. The measurement descriptor contents include the current RIM value. The new RIM value is computed by applying the RHA to the measurement descriptor.

$$desc = MeasurementDescriptor(M_{i-1}, ...)$$
$$M_i = RHA(desc)$$

DRAFT

I_{FQHFC} A RIM is immutable while the state of the Realm is REALM_ACTIVE. This implies that a RIM reflects the configuration and contents of the Realm at the moment when it transitioned from the REALM_NEW to the REALM_ACTIVE 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 RSI_MEASUREMENT_READ command.

See also:

- B4.5.37.4 *RMI_REALM_CREATE initialization of RIM*
- B4.5.40.4 *RMI_REC_CREATE extension of RIM*
- B4.5.58.4 *RMI_RTT_DATA_MAP_INIT extension of RIM*
- B5.4.9 *RSI_MEASUREMENT_READ command*

A7.1.2 Realm Extensible Measurement

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

I_{QJDWM} A REM is extended using the RSI_MEASUREMENT_EXTEND command.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- B5.4.8 *RSI_MEASUREMENT_EXTEND command*
- B5.4.9 *RSI_MEASUREMENT_READ command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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, the CCA platform on which the Realm is running, and the set of devices assigned to the Realm.

I_{BXBSD} A CCA attestation token consists of three parts:

- Realm token

  Contains attributes of the Realm, including:

  - Realm Initial Measurement
  - Realm Extensible Measurements

- Device token

  Contains a list of devices assigned to the Realm

- 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 the RSI Granule size.

See also:

- A7.1.1 *Realm Initial Measurement*
- A7.1.2 *Realm Extensible Measurement*

DRAFT

A7.2.2 Attestation token generation

I_{KRMRH} 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

S_{PNCRK} 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*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A7.2.3.1.1 *Realm challenge claim*
- B5.4.2 *RSI_ATTESTATION_TOKEN_CONTINUE command*
- B5.4.3 *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

I_{TFHGX} The CCA attestation token is a profiled IETF Entity Attestation Token (EAT).

I_{LPTVH} The CCA attestation token is structured as a Conceptual Messages Wrapper (CMW) envelope *RATS Conceptual Messages Wrapper (CMW)* [7]. The CMW is a CBOR map that contains the CCA platform token and the Realm token, each encoded as an EAT CWT, and optionally contains a device token.

I_{YZPHG} The Realm token contains structured data in CBOR, wrapped with a COSE_Sign1 envelope according to the CBOR Object Signing and Encryption (COSE) standard.

I_{MMQZG} The Realm token is signed by the Realm Attestation Key (RAK).

I_{SNLKT} The CCA device token is an unsigned CBOR structure that represents the set of devices assigned to a Realm.

I_{WBGNP} The CCA platform token contains structured data in CBOR, wrapped with a COSE_Sign1 envelope according to the COSE standard.

I_{CGYKX} The CCA platform token is signed by the CCA Platform Attestation Key (CPAK).

I_{LMXMJ} Where the CPAK is endorsed via an X.509 certificate chain, the endorsement artefacts can be included in the COSE_Sign1 envelope of the CCA platform token using parameters from *CBOR Object Signing and Encryption (COSE) Header Parameters for Carrying and Referencing X.509 Certificates* [8]. It is recommended that this is done as follows:

- The CPAK certificate is identified by including an `x5t` thumbprint parameter in the COSE_Sign1 protected header.
- The CPAK certificate itself is then packaged within an `x5chain` parameter in the COSE_Sign1 unprotected header.
- This `x5chain` parameter can also include other certificates that endorse the CPAK certificate.

Alternatively, the other certificates in the chain that endorses the CPAK certificate can be packaged in an additional entry within the *RATS Conceptual Messages Wrapper (CMW)* [7] token.

I_{CCGQH} The CCA platform token contains a hash of RAK_pub. This establishes a cryptographic binding between the Realm token and the CCA platform token.

I_{PTKYD} The CCA attestation token is defined as follows:

```
;# import rfc9052

; CBOR-tagged CMW Collection, see draft-ietf-rats-msg-wrap
cca-token = #6.907(cca-token-CMW)

EAT_CWT = (
    bstr .cbor  COSE_Sign1_Tagged
)

CMW-CWT-element = (
    [
        eat-cwt-coap-label,
        EAT_CWT
    ]
)

cca-device-token-bstr = (
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    bstr .cbor cca-device-claims
)

CMW-UCCS-device = (
    [
        eat-uccs-coap-label,
        cca-device-token-bstr
    ]
)

cca-token-CMW = {
    cca-platform-token-label =>  CMW-CWT-element,
    cca-realm-delegated-token-label =>  CMW-CWT-element,
    ? cca-device-token-label =>  CMW-UCCS-device
}

cca-platform-token-label = 44234
cca-realm-delegated-token-label = 44241
cca-device-token-label = 44258
eat-cwt-coap-label = 263                              ; application/eat+cwt

; the following labels are placeholders to be used where an implementation
;  requires additional CMW entries
cca-realm-direct-token-label = 44251
eat-uccs-coap-label = 601                             ; application/uccs+cbor

cca-platform-selective-device-evidence = 44252
; device evidence is either uccs+cbor if hash locked or eat+cwt if signed

cca-platform-cpak-cert-chain-label = 44254
; cert-chain-coap-label = TBD                         ; application/pkix-pkipath
cca-platform-cpak-cert-chain-with-evidence-label = 44255
; cert-chain-with-evidence-coap-label = TBD          ; application/dice-chain

cca-platform-firmware-activity-log-label = 44256
; activity-log-coap-label = TBD                       ; format IMPDEF
```

I_{HZNNH} The composition of the CCA attestation token is summarised in the following figure.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

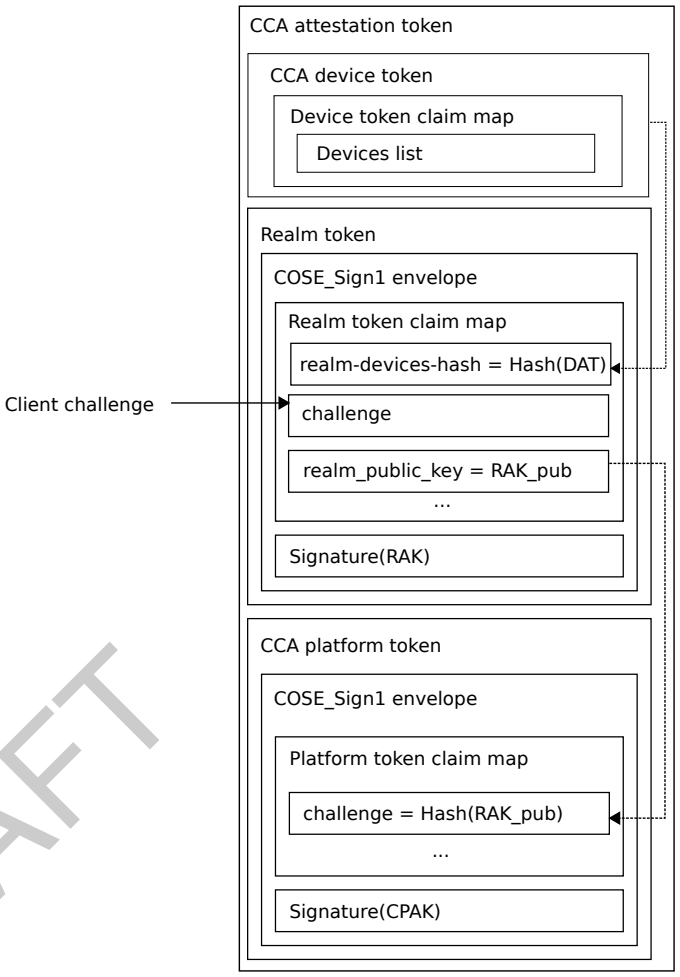


Figure A7.1: Attestation token format

DRAFT

See also:

- *Arm CCA Security model* [4]
- *Concise Binary Object Representation (CBOR)* [9]
- *CBOR Object Signing and Encryption (COSE)* [10]
- *Entity Attestation Token (EAT)* [11]
- A7.2.3.1 *Realm claims*
- A7.2.3.2 *CCA device claims*
- A7.2.3.3 *CCA platform claims*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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-instance-id
    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-lfa-policy
    ? cca-realm-devices-token-hash
}
```

See also:

- *Concise Data Definition Language (CDDL)* [12]
- A7.2.3.1.1 *Realm challenge claim*
- A7.2.3.1.2 *Realm profile claim*
- A7.2.3.1.3 *Realm Instance ID claim*
- A7.2.3.1.4 *Realm Personalization Value claim*
- A7.2.3.1.5 *Realm Initial Measurement claim*
- A7.2.3.1.6 *Realm Extensible Measurements claim*
- A7.2.3.1.7 *Realm hash algorithm ID claim*
- A7.2.3.1.8 *Realm MEC policy claim*
- A7.2.3.1.9 *Realm LFA policy claim*
- A7.2.3.1.10 *Realm public key claim*
- A7.2.3.1.11 *Realm public key hash algorithm identifier claim*
- A7.2.3.1.12 *Realm devices token hash claim*
- A7.2.3.1.13 *Collated CDDL for Realm claims*
- A7.2.3.1.14 *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:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A7.2.2 *Attestation token generation*
- B5.4.3 *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#2.0.0"

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

A7.2.3.1.3 Realm Instance ID claim

I_{YSHVT} The Realm Instance ID claim is a random value generated using a cryptographic-quality entropy source.

I_{YHLZK} The Realm Instance ID claim is identified using the EAT `ueid` label (256).

R_{BVCXM} The first byte of the Realm Instance ID value must be `0x01`.

I_{TFZYD} The Realm Instance ID claim must be present in a Realm token.

I_{LTZVS} The format of the CCA Realm Instance ID claim is defined as follows:

```
cca-realm-instance-id-label = 256 ; EAT ueid

; EAT UEIDs need to be 7 - 33 bytes
cca-realm-instance-id-type = bytes .size 33

cca-realm-instance-id = (
    cca-realm-instance-id-label => cca-realm-instance-id-type
)
```

See also:

- *Arm CCA Security model* [4]

A7.2.3.1.4 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:

- A2.2.3 *Realm attributes*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.1.5 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*
- A7.2.3.1.6 *Realm Extensible Measurements claim*

A7.2.3.1.6 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*
- A7.2.3.1.5 *Realm Initial Measurement claim*

A7.2.3.1.7 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* [13].

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
)
```

A7.2.3.1.8 Realm MEC policy claim

I_{RWPDL} The Realm MEC policy identifies the MEC policy of the Realm.

I_{YBMNH} The Realm MEC policy claim must be present in a Realm token.

R_{WRXDS} On a platform which does not implement FEAT_MEC, the value of the Realm MEC policy claim is cca-realm-mec-policy-shared.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{FHVCV} The format of the Realm MEC policy claim is defined as follows:

```
cca-realm-mec-policy-label = 44243

cca-realm-mec-policy-shared = 0
cca-realm-mec-policy-private = 1

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

See also:

- Chapter A11 *Realm memory encryption*

A7.2.3.1.9 Realm LFA policy claim

I_{MWPNQ} The Realm LFA policy identifies the Live Firmware Activation policy of the Realm.

I_{BCPFQ} The Realm LFA policy claim is optional in a Realm token.

I_{HLHSP} The format of the Realm LFA policy claim is defined as follows:

```
cca-realm-lfa-policy-label = 44244

cca-realm-lfa-policy-disallow = 0
cca-realm-lfa-policy-allow = 1

cca-realm-lfa-policy = (
    cca-realm-lfa-policy-label =>
        cca-realm-lfa-policy-disallow /
        cca-realm-lfa-policy-allow
)
```

See also:

- A3.13 *Live Firmware Activation*

A7.2.3.1.10 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:

```
;# import rfc9052

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
)
```

See also:

- *SEC 1: Elliptic Curve Cryptography, version 2.0* [14]
- A7.2.3.1.11 *Realm public key hash algorithm identifier claim*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A7.2.3.3.2 *CCA platform challenge claim*

A7.2.3.1.11 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* [14]
- A7.2.3.1.10 *Realm public key claim*
- A7.2.3.3.2 *CCA platform challenge claim*

A7.2.3.1.12 Realm devices token hash claim

I_{MTZSZ} The Realm devices token hash claim contains a hash of the CBOR encoding of cca-realm-devices-token that represents the set of devices assigned to the Realm.

I_{HWQFR} The Realm devices token hash claim must be present in a Realm token whenever one or more devices are assigned to the Realm. If no devices are assigned, the claim must not be present.

I_{RYZLN} The format of the Realm devices token hash claim is defined as follows:

```
cca-realm-devices-token-hash-label = 44257

cca-realm-devices-token-hash = (
    cca-realm-devices-token-hash-label => cca-hash-type
)
```

See also:

- A7.2.3.2 *CCA device claims*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.1.13 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-instance-id
    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-lfa-policy
    ? cca-realm-devices-token-hash
}
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#2.0.0"

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
)
;# import rfc9052

cca-realm-public-key-label = 44237

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


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
cca-realm-public-key = (
    cca-realm-public-key-label => cca-realm-public-key-type
)
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 = 44243

cca-realm-mec-policy-shared = 0
cca-realm-mec-policy-private = 1

cca-realm-mec-policy = (
    cca-realm-mec-policy-label =>
        cca-realm-mec-policy-shared /
        cca-realm-mec-policy-private
)
cca-realm-lfa-policy-label = 44244

cca-realm-lfa-policy-disallow = 0
cca-realm-lfa-policy-allow = 1

cca-realm-lfa-policy = (
    cca-realm-lfa-policy-label =>
        cca-realm-lfa-policy-disallow /
        cca-realm-lfa-policy-allow
)
cca-hash-type = bytes .size 32 / bytes .size 48 / bytes .size 64
cca-realm-devices-token-hash-label = 44257

cca-realm-devices-token-hash = (
    cca-realm-devices-token-hash-label => cca-hash-type
)
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.1.14 Example Realm claims

I_{CPTFR} An example Realm claim map is shown below in COSE-DIAG format:

```
/ Realm claim map /
{
    / cca-realm-profile /
    265: "tag:arm.com,2024:realm#2.0.0",

    / cca-realm-challenge /
    10: h'ABABABABABABABABABABABABABABABAB
          ABABABABABABABABABABABABABABABAB
          ABABABABABABABABABABABABABABABAB
          ABABABABABABABABABABABABABABABAB',

    / cca-realm-instance-id /
    256: h'010BBBBBBBBBBBBBBBBBBBBBBBBBBBB
           BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
           BB',

    / cca-realm-personalization-value /
    44235: h'ABABABABABABABABABABABABABABABAB
             ABABABABABABABABABABABABABABABAB
             ABABABABABABABABABABABABABABABAB
             ABABABABABABABABABABABABABABABAB',

    / cca-realm-initial-measurement /
    44238: h'000000000000000000000000000000000
             000000000000000000000000000000000',

    / cca-realm-extensible-measurements /
    44239: [
        h'000000000000000000000000000000000
          000000000000000000000000000000000',
        h'000000000000000000000000000000000
          000000000000000000000000000000000',
        h'000000000000000000000000000000000
          000000000000000000000000000000000',
        h'000000000000000000000000000000000
          000000000000000000000000000000000'
    ],

    / cca-realm-hash-algo-id /
    44236: "sha-256",

    / cca-realm-public-key /
    44237: h'A501020338232002215820 66EEA6A226
             78C3A9F83148EF349800B20ABB486F2C
             C6D7ED017EC49798C8D4372258202F25
             DE86812374E6E8D48DEE8E230AD29CCD
             839BE6E0DB8C7AB9DEDE0805D29D',

     / cca-realm-public-key-hash-algo-id /
     44240: "sha-256",

     / cca-realm-mec-policy /
     44243: 0,

     / cca-realm-lfa-policy /
     44244: 1,
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        / cca-realm-devices-token-hash /
        44257: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
}
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.2 CCA device claims

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

I_{BPBTP} The CCA device token claim map is defined as follows:

```
; CCA device token
cca-device-claims = (cca-device-claim-map)

cca-device-claim-map = {
    cca-device-profile
    cca-devices
}
```

See also:

- *Concise Data Definition Language (CDDL)* [12]
- A7.2.3.2.1 *CCA device profile claim*
- A7.2.3.2.2 *CCA devices claim*
- A7.2.3.2.3 *Collated CDDL for CCA device claims*
- A7.2.3.2.4 *Example Device claims*

A7.2.3.2.1 CCA device profile claim

I_{ZFFXM} The CCA device profile claim identifies the EAT profile to which the CCA device token conforms.

I_{NFJMF} The CCA device profile claim is identified using the EAT `profile` label (265).

I_{CQVZC} The CCA device profile claim must be present in a CCA device token.

I_{PVJTS} The format of the CCA device profile claim is defined as follows:

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

cca-device-profile-type = "tag:arm.com,2025:cca_device#2.0.0"

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

A7.2.3.2.2 CCA devices claim

I_{FZMBB} The CCA device identity digest is a cca-hash-type value which represents the digest of the device's identity.

I_{QHWQB} The CCA device measurement digest is a cca-hash-type value representing a digest of the device's measurement evidence. If communication between the RMM and the device uses SPDM, the digest of the device measurements are the SPDM measurements exchange, request and response, between the device and the RMM. If communication between the RMM and the device uses an IMPLEMENTATION DEFINED channel, the digest of the device measurements contains a digest of the received measurements from the device.

I_{GSVNY} The CCA device protocol negotiation data digest is a cca-hash-type value which represents the digest of the communication protocol negotiation data. If communication between the RMM and the device uses SPDM, this field corresponds to the VCA digest.

I_{WRLZT} The CCA devices claim must be present in a CCA device token.

I_{PPKQK} The CCA device coherent traffic IDE protection is a boolean value which specifies whether coherent traffic between the host and the device is protected by IDE.

I_{CDMFD} The CCA device coherent traffic IDE protection field must be present in a CCA device claim.

I_{YXYYR} The CCA device non-coherent traffic IDE protection is a boolean value which specifies whether non-coherent traffic between the host and the device is protected by IDE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{HJSXQ} The CCA device non-coherent traffic IDE protection field must be present in a CCA device claim.

I_{ZFPLT} Devices are listed in the CCA devices claim in the order with which the corresponding VDEVs were created.

I_{CBLCX} The format of the CCA devices claim is defined as follows:

```
cca-devices-label = 44259

cca-device = {
    1 => cca-hash-type,     ; device identity digest
    2 => cca-hash-type,     ; device measurements exchange digest
    ? 3 => cca-hash-type,   ; protocol negotiation data digest
    4 => bool,              ; coherent traffic IDE protection
    5 => bool,              ; non-coherent traffic IDE protection
}

cca-devices = (
    cca-devices-label => [ + cca-device ]
)
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.2.3 Collated CDDL for CCA device claims

D_{CZRNS} The format of the CCA device token claim map is defined as follows:

```
; CCA device token
cca-device-claims = (cca-device-claim-map)

cca-device-claim-map = {
    cca-device-profile
    cca-devices
}
cca-device-profile-label = 265 ; EAT profile

cca-device-profile-type = "tag:arm.com,2025:cca_device#2.0.0"

cca-device-profile = (
    cca-device-profile-label => cca-device-profile-type
)
cca-hash-type = bytes .size 32 / bytes .size 48 / bytes .size 64
cca-devices-label = 44259

cca-device = {
    1 => cca-hash-type,     ; device identity digest
    2 => cca-hash-type,     ; device measurements exchange digest
    ? 3 => cca-hash-type,   ; protocol negotiation data digest
    4 => bool,              ; coherent traffic IDE protection
    5 => bool,              ; non-coherent traffic IDE protection
}

cca-devices = (
    cca-devices-label => [ + cca-device ]
)
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.2.4 Example Device claims

I_{MNMGY} An example device claim map is shown below in DIAG format:

```
/ Device claim map /
{
    / cca-device-profile /
    265: "tag:arm.com,2025:cca_device#2.0.0",

    / cca-devices /
    44259: [
        {
          / device identity digest /
          1: h'000102030405060708090A0B0C0D0E0F
               101112131415161718191A1B1C1D1E1F',

          / device measurements digest /
          2: h'202122232425262728292A2B2C2D2E2F
               303132333435363738393A3B3C3D3E3F',

          / protocol negotiation data digest /
          3: h'404142434445464748494A4B4C4D4E4F
               505152535455565758595A5B5C5D5E5F',

          / coherent traffic IDE protection /
          4: false,

          / non-coherent traffic IDE protection /
          5: true
        }
      ]
}
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.3 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
    ? cca-platform-manufacturing-config
    cca-platform-client-id
}
```

See also:

- *Concise Data Definition Language (CDDL)* [12]
- A7.2.3.3.1 *CCA platform profile claim*
- A7.2.3.3.2 *CCA platform challenge claim*
- A7.2.3.3.3 *CCA platform Implementation ID claim*
- A7.2.3.3.4 *CCA platform Instance ID claim*
- A7.2.3.3.5 *CCA platform config claim*
- A7.2.3.3.6 *CCA platform manufacturing config claim*
- A7.2.3.3.7 *CCA platform lifecycle claim*
- A7.2.3.3.8 *CCA platform software components claim*
- A7.2.3.3.9 *CCA platform verification service claim*
- A7.2.3.3.10 *CCA platform hash algorithm ID claim*
- A7.2.3.3.11 *CCA platform client ID claim*
- A7.2.3.3.12 *Collated CDDL for CCA platform claims*
- A7.2.3.3.13 *Example CCA platform claims*

A7.2.3.3.1 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_{GMKNR} 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#2.0.0"

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

A7.2.3.3.2 CCA platform challenge claim

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

I_{XHLYJ} 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:

- A7.2.3.1.10 *Realm public key claim*

A7.2.3.3.3 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]
- A7.2.3.3.4 *CCA platform Instance ID claim*

A7.2.3.3.4 CCA platform Instance ID claim

I_{ZYRZB} The CCA platform Instance ID claim represents the unique identifier of the CCA Platform Attestation Key (CPAK) 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

; EAT UEIDs need to be 7 - 33 bytes
cca-platform-instance-id-type = bytes .size 33
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

See also:

- *Arm CCA Security model* [4]
- A7.2.3.3.3 *CCA platform Implementation ID claim*

A7.2.3.3.5 CCA platform config claim

I_{WVQJT} 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.

U_{GPXWX} The CCA platform config byte string contains implementation information that is provided by the chip vendor and the device vendor. This is expected to include the following system properties:

- Per-PAS encryption (all RME systems will require this property)
- MEC
- MPE Level
  - L0 (none)
  - L1 (encryption only)
  - L2 (encryption and integrity)
  - L3 (anti-replay)
- RME-DA support
- RME-CDA support

The layout and encoding of this information is IMPLEMENTATION DEFINED.

An attestation verifier should use information from the relevant attestation profile document to understand the IMPLEMENTATION DEFINED choices made for this field.

I_{MJHQJ} 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* [15]

A7.2.3.3.6 CCA platform manufacturing config claim

I_{RQKNX} The CCA platform manufacturing config claim represents a record of production phases and testing conducted during the manufacturing process for this instance.

I_{CJRBF} The CCA platform manufacturing config claim is optional in a CCA platform token.

I_{FYFPJ} The format of the CCA platform manufacturing config claim is defined as follows:

```
cca-platform-manufacturing-config-label = 2403
cca-platform-manufacturing-config-type = bytes

cca-platform-manufacturing-config = (
    cca-platform-manufacturing-config-label =>
        cca-platform-manufacturing-config-type
)
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.3.7 CCA platform lifecycle claim

I_{SYKFY} The CCA platform lifecycle claim identifies the lifecycle state of the CCA platform.

R_{NBFVV} The value of the CCA platform lifecycle claim is an integer which is divided as follows:

- value[15:8]: CCA platform lifecycle state
- value[7:0]: IMPLEMENTATION DEFINED

I_{WFZHV} The CCA platform lifecycle claim must be present in a CCA platform token.

I_{QFYLF} 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.

I_{HMZLL} 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]

A7.2.3.3.8 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.

U_{VDQYM} 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
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    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.

U_{JJLJT} 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. The measurement values are implemented such that the values can only be extended rather than set. The values are initialised to 0, so the value reported in attestation will be `H( 0 || H(software component))`. If the CCA platform supports Live Firmware Activation then the value reported in attestation may have been further extended by measurements of updates to the software component. In this case, the value of the measurement must be validated by reconstructing the reported value using information from the Firmware Activity Log.

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 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* [13].

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

I_{YYTVP} 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.

I_{LWQHQ} 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

I_{MBBKC} 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.

U_{WXFDW} 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

U_{ZCPJJ} 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.

I_{FVDZF} The CCA platform software component countersigner ID list is optional in a CCA platform token.

A7.2.3.3.9 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
)
```

A7.2.3.3.10 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* [13].

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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
)
```

A7.2.3.3.11 CCA platform client ID claim

I_{PTXFD} The CCA platform client ID claim identifies the security domain from which the attestation token was requested.

I_{KBVSR} In this version of the specification, the only valid value for the CCA platform client ID claim is the Realm Management Security Domain (RMSD).

I_{VYFPY} The CCA platform client ID claim must be present in a CCA platform token.

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

```
cca-platform-client-id-label = 2394 ; PSA namespace

cca-platform-client-id-rmsd = 1 ; Realm Management Security Domain

cca-platform-client-id-type =
    cca-platform-client-id-rmsd

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.3.12 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-manufacturing-config
    cca-platform-client-id
}
cca-platform-profile-label = 265 ; EAT profile

cca-platform-profile-type = "tag:arm.com,2024:cca_platform#2.0.0"

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 ueid

; 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
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
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
)
cca-platform-client-id-label = 2394 ; PSA namespace

cca-platform-client-id-rmsd = 1 ; Realm Management Security Domain

cca-platform-client-id-type =
    cca-platform-client-id-rmsd

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


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A7.2.3.3.13 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#2.0.0",

    / cca-platform-challenge /
    10: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
          AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
          AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
          AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',

    / cca-platform-implementation-id /
    2396: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
            AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',

    / cca-platform-instance-id /
    256: h'010BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
           BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
           BB',

    / cca-platform-config /
    2401: h'CFCFCFCF',

    / cca-platform-manufacturing-config /
    2403: h'ABABABAB',

    / cca-platform-lifecycle /
    2395: 12288,

    / cca-platform-sw-components /
    2399: [
        {
            / measurement value /
            2: h'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA',

            / signer id /
            5: h'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
                 BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
                 BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
                 BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB',

            / version /
            4: "1.0.0",

            / hash algorithm identifier /
            6: "sha-256"
        },
        {
            / measurement value /
            2: h'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
                 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
                 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
                 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC',
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
                / signer id /
                5: h'DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
                     DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
                     DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
                     DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD',

                / 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",

        / cca-platform-client-id /
        2394: 1

    }
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A8

Realm debug and performance monitoring

DRAFT

This section describes the debug and performance monitoring features which are available to a Realm.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A8.1 Realm PMU

This section describes the programming model for usage of PMU by a Realm.

D_{NZKTL} *Realm-assigned PMU counters* means PMU counters in the range `[0 .. realm.num_pmu_ctrs)`.

I_{HQPJR} Host state for Realm-assigned PMU counters is not guaranteed to be preserved during Realm execution.

R_{DNNQQ} On REC entry, the state of all Realm-assigned PMU counters is restored from the REC object.

R_{LHRYJ} On REC exit, the state of all Realm-assigned PMU counters is saved to the REC object.

R_{WXTZF} On REC exit, `exit.pmu_ovf_status` indicates the level of the PMU overflow interrupt which results from the state of the Realm-assigned PMU counters.

U_{MCKDV} Realm access to the following registers should be emulated by the RMM.

- `PMCR_EL0`
- `PMINTEN{CLR,SET}_EL0`
- `PMOVS{CLR,SET}_EL0`

U_{PKBFG} On REC exit with `exit.pmu_ovf_status == RMI_PMU_OVERFLOW_ACTIVE`, the Host is not required to change the state of the physical PMU overflow interrupt before the next REC entry. Consequently, the RMM should disable overflow interrupts for Realm-assigned PMU counters on Realm entry, to avoid an active physical PMU overflow interrupt resulting in an immediate Realm exit.

D_{RQHVG} *Host-retained PMU counters* means PMU counters in the range `[realm.num_pmu_ctrs .. 31)`.

R_{JSXJM} During Realm execution, `PMCNTENSET_EL0.C` behaves according to its architectural specification. If the Host has enabled the processor Cycle Counter, it continues to count during Realm execution.

R_{XZBJD} During Realm execution, `PMCNTENSET_EL0.P<m>` behaves as `0` for all Host-retained PMU counters. All Host-retained PMU counters are disabled during Realm execution.

I_{DSNZD} The following diagram illustrates how overflows are handled for Realm-assigned PMU counters.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

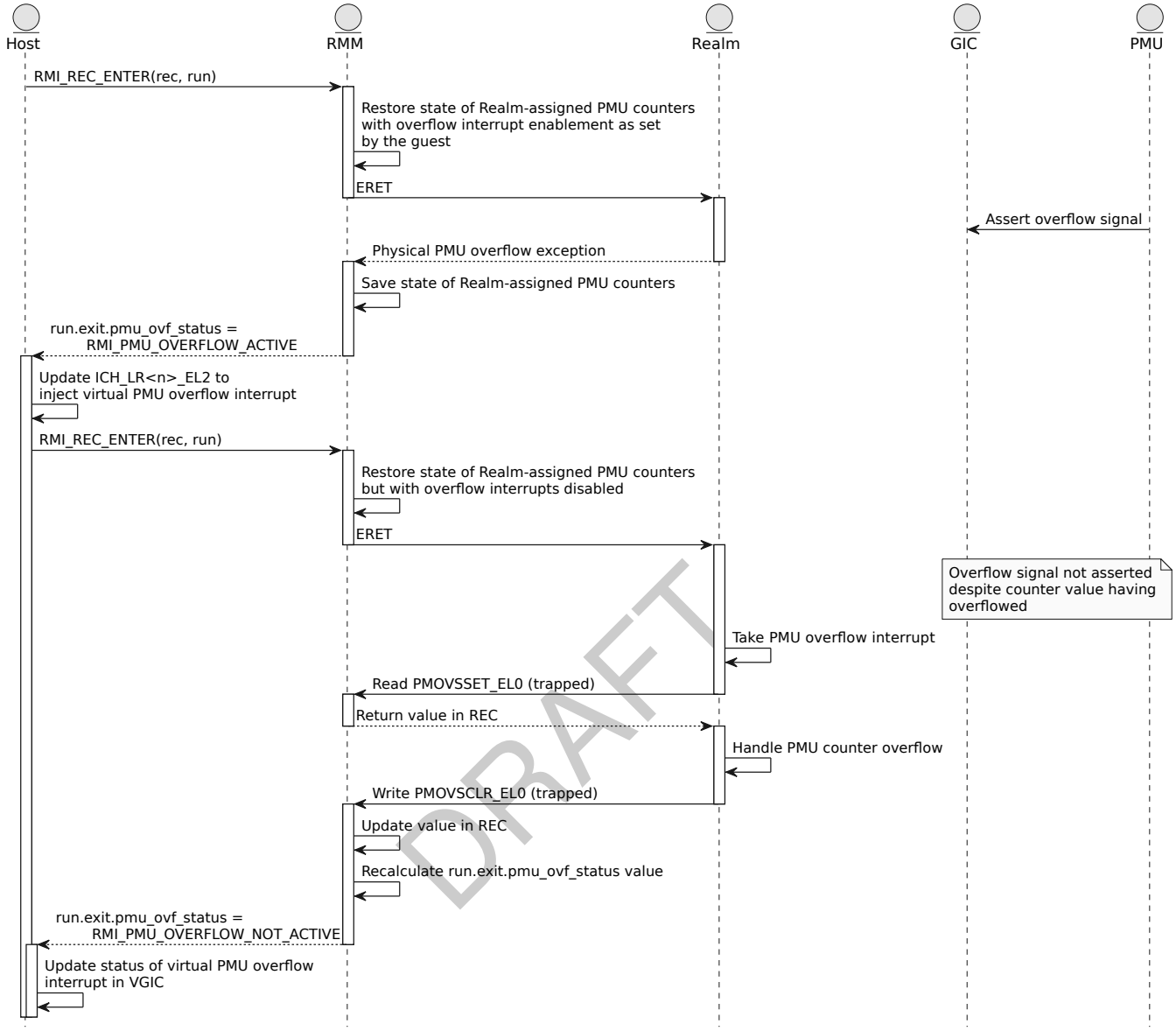


Figure A8.1: Realm PMU overflow

See also:

- A3.6 *Realm support for Performance Monitors Extension*
- A4.3 *REC exit*
- B4.6.66 *RmiRecExit type*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A9
Realm device assignment

DRAFT

This section describes how devices are assigned to Realms, attested and granted permission to access Realm-owned memory.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.1 Realm device assignment overview

I_{DKBCY} 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.

A9.1.1 Device objects

I_{WQVRH} 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.

- *Physical SMMU* (PSMMU)

  Allows the RMM to apply the same stage 2 translations to Realm-assigned devices as for Realm VPEs.

- *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 inititated by assigned device functions.

Interaction between the Host and a PDEV, VDEV and VSMMU objects is performed via RMI commands.

See also:

- A9.2 *Physical device object*
- A9.4 *Virtual device object*
- A9.7 *Physical SMMU*
- A9.8 *Virtual SMMU*

DRAFT

A9.1.2 Device properties

D_{PTXBK} The trust model of a PDEV determines the actions which are necessary for the whole device, or one of its interfaces, to be admitted into the TCB of a Realm. A *device trust model* is one of the following:

- *Selective trust*: a Realm makes a decision whether to trust a given VDEV which has been assigned to it. Until and unless the Realm informs the RMM that it trusts the VDEV, the device is not granted access to the Realm's memory.

D_{GRWDX} The communication model of a PDEV determines how the RMM communicates management requests to the device. The communication model also constrains the form of device identity evidence used during device assignment. A *device communication model* is one of the following:

- *SPDM communication*: communication consists of Security Protocol and Data Model (SPDM) messages, transported via Non-secure memory.
  - *Device identity evidence*: the device certificate chain.
- *Platform communication*: communication is performed via a secure IMPLEMENTATION DEFINED channel.
  - *Device identity evidence*: IMPLEMENTATION DEFINED.

D_{BKVYM} The traffic protection model of a PDEV determines how the confidentiality and integrity of traffic between the SoC and the device is protected. A *device traffic protection model* is one of the following:

- *IDE protection*: traffic is protected using the Integrity and Data Encryption (IDE) standard.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- *Platform protection*: traffic is protected via system construction, for example by restricting physical access to the transport.

Coherent and non-coherent traffic to a given device may be subject to different traffic protection models.

See also:

- *PCI Express 6.0 specification* [16]
- *Secured Messages using SPDM Specification version 1.1.0* [17]
- A9.5 *Communication between RMM and a device*

A9.1.3 Device assignment flow

A9.1.3.1 Assignment of a selective-trust device

I_SZYSK Assignment of a selective-trust device to a Realm involves the following steps:

1. The Host creates and initializes a PDEV object, associated with the target physical device. This causes the following to happen:
   - A secure communication channel is established between the RMM and the device. Details depend on the *device communication model*.
   - Protection is configured for traffic between the SoC and the device. Details depend on the *device traffic protection model*.
   - The device identity evidence 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 identity evidence is stored by the RMM. This is used later to check integrity of the attestation evidence provided by the Host to the Realm.
2. If communication between the RMM and the device uses SPDM, the Host extracts the public key from the device certificate chain and provides it to the RMM. The RMM verifies that the device to which it has a secure communication channel holds the corresponding private key. The RMM stores a digest of the public key.
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. Optionally, 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.

   Alternatively, the Host can create these mappings later, in response to a request from the guest.
5. The Host requests the RMM to retrieve device attestation evidence (device interface report and device measurements.) The RMM provides the attestation evidence to the Host for caching, and the RMM stores digests of the attestation evidence.

   The Realm later requests device attestation evidence from the Host, and verifies that this matches the corresponding digests stored by the RMM.
6. If communication between the RMM and the device uses SPDM, the Realm verifies that the digest of the public key held by the RMM matches the public key in the device certificate chain.
7. The Realm verifies that the negotiation data, device identity evidence, and device measurements are acceptable.
8. The Realm requests the RMM to verify 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.
9. The Realm instructs the RMM to grant the device access to Realm-owned memory.

DRAFT

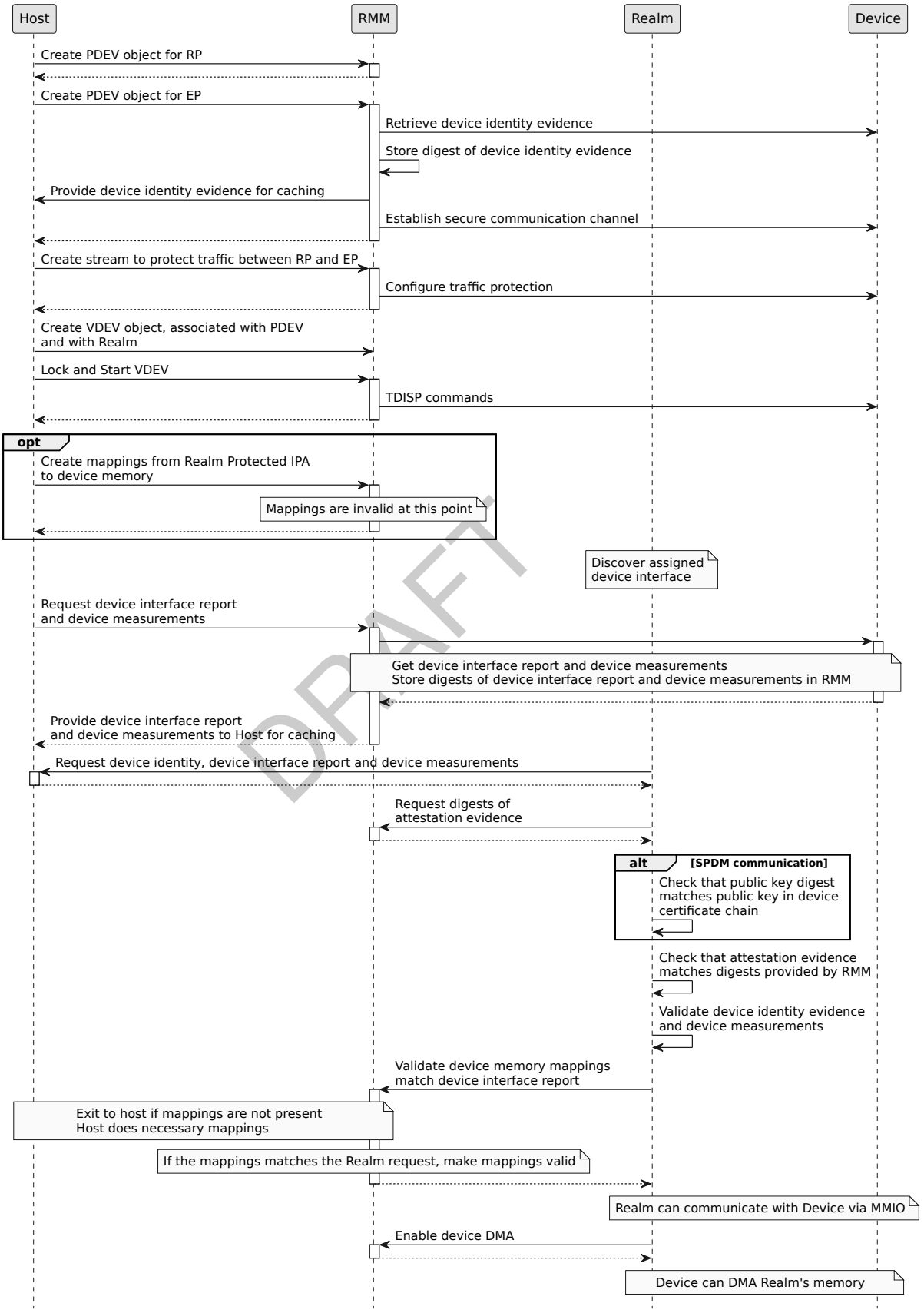
Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

X_{RSSXY} 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.

I_{RBJHR} Assignment of a selective-trust device to a Realm is illustrated in the following sequence diagram.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



DRAFT

Figure A9.1: Assignment of a selective-trust device to a Realm

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{PYHYC} For a selective-trust device which uses SPDM communication, all of the following are true:

- The device certificate chain digest computed by the RMM 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.2 Physical device object

D_{LHTXR} A *Physical Device* (PDEV) represents a communication channel between the RMM and a physical device.

I_{HDVMG} The physical device represented by a PDEV is one of the following:

- A Root Port.
- An endpoint device whose resources are mapped as special-purpose memory. For example, a PCIe accelerator or a CHI / CXL accelerator.

A9.2.1 Physical device attributes

D_{WKMVW} The attributes of a PDEV are summarized in the following table.

Name	Type	Description
category	RmmPdevCategory	Device category
pdev_id	Bits64	Device identifier
routing_id	Bits64	Routing identifier
rid_base	Bits32	Base of requester ID range (inclusive). For PCIe category of devices, the value is in BDF format.
rid_top	Bits32	Top of requester ID range (exclusive). For PCIe category of devices, the value is in BDF format.
spdm	RmmPdevSpdm	Whether communication with the device uses SPDM
signed_meas	RmmFeature	Whether device supports signed measurements
id_index	UInt64	Device identity index
hash_algo	RmmHashAlgorithm	Algorithm used to generate device digests
state	RmmPdevState	Lifecycle state
op	RmmPdevOperation	Operation performed on this PDEV
comm_state	RmmDevCommState	Device communication state
max_num_vdevs	UInt64	Maximum number of VDEVs which can be associated with this PDEV
num_vdevs	UInt64	Number of VDEVs associated with this PDEV
protocol_data_diges	Bits512	Protocol data digest

I_{DGWSB} For a device which uses SPDM communication, protocol_data_digest field corresponds to the VCA digest. VCA is a concatenation of the following SPDM requests and responses:

- VERSION_REQ
- VERSION_RESP
- CAP_REQ
- CAP_RESP
- NEGO_REQ
- NEGO_RESP

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_JJSKC The following table describes how combinations of PDEV attributes map to different types of devices.

PDEV attributes		
spdm	**category**	**Device type**
1	ENDPOINT_ROOT_PORT	Root Port
1	ENDPOINT_ACCEL_OFF_CHIP	Off-chip PCIe accelerator
*	ENDPOINT_ACCEL_ON_CHIP	Integrated PCIe device
1	ENDPOINT_ACCEL_OFF_CHIP	Off-chip coherent accelerator
*	ENDPOINT_ACCEL_ON_CHIP	Integrated coherent accelerator

- * indicates that either 0 or 1 is permitted.

I_0001 The following table describes the validity of the various fields in PDEV params depending on the different categories of devices.

Fields	**ENDPOINT_ACCEL_OFF_CHIP**	**ENDPOINT_ACCEL_OFF_CHIP**	**ROOT_PORT**
flags	Valid	Valid	Valid
pdev_id	Valid	Valid	Valid
routing_id	Valid	Valid	Valid
id_index	Valid	Valid	Ignored
rid_base	Valid	Valid	Ignored
rid_top	Valid	Valid	Ignored
hash_algo	Valid	Valid	Ignored
max_vdevs_order	Valid	Valid	Ignored

DRAFT

I_RQCKC The following table summarizes the meaning of the pdev_id and routing_id attributes, for each supported device category.

category	**pdev_id**	**routing_id**
ENDPOINT_ACCEL_OFF_CHIP or ENDPOINT_ACCEL_ON_CHIP or ROOT_PORT	Address of device entry in PCIe configuration space. The ECAM physical (system) address - the unique CPU-visible system physical address to the start of the target's or RP's PCIe Extended Configuration Space within an ECAM MMIO region.	PCIe segment identifier.

See also:

- *PCI Express 6.0 specification* [16]
- *Compute eXpress Link specification* [18]
- B3.120 *RmiPdevFlagsSupported function*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.2.2 Physical device invariants

This section lists invariants which are enforced via checks performed by the RMM on execution of RMI_PDEV_CREATE.

R_{YDSRN} On execution of RMI_PDEV_CREATE, TDISP_EN=1 is set at the corresponding Root Port.

U_{YJXZG} The latest point when the implementation may set TDISP_EN=1 is on creation of the first PDEV attached to that Root Port.

R_{DFZJC} pdev.max_num_vdevs is not greater than ((2 ^ rmm.static.max_vdevs_order) - 1).

R_{JZKQM} Once the state of a PDEV has transitioned to PDEV_READY, the RMM has established secure communication with the device, which remains in place until the PDEV transitions to PDEV_STOPPED.

R_{RZLHW} For an integrated PDEV the range (pdev.rid_base, pdev.rid_top] falls within the platform-assigned RID range for the corresponding device.

See also:

- A2.3.2 *Views of physical memory*
- A2.3.4 *Granule tracking region*
- A9.4.2 *Virtual device invariants*
- B4.5.18 *RMI_PDEV_CREATE command*

A9.2.3 Physical device lifecycle

A9.2.3.1 States

D_{GJRWJ} 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.
PDEV_STOPPED	Secure connection between the RMM and the device has been terminated.
PDEV_ERROR	Device has reported a fatal error.

A9.2.3.2 State transitions

I_{WRZQD} 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.

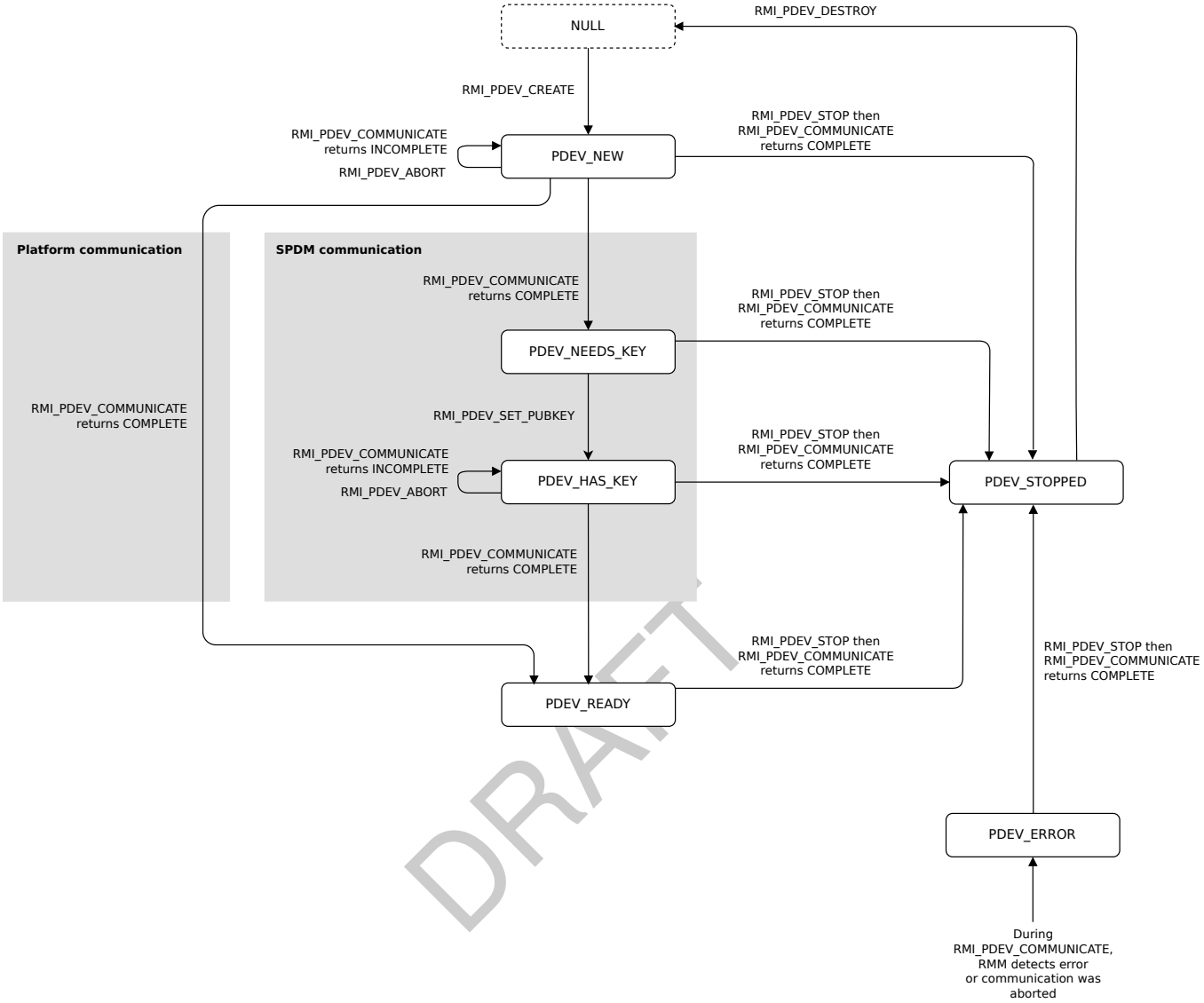
A transition from the pseudo-state *NULL* represents creation of a PDEV. A transition to the pseudo-state *NULL* represents destruction of a PDEV.

In this diagram:

- “RMI_PDEV_COMMUNICATE returns COMPLETE” means that the RmiDevCommExitFlags indicate that device communication is complete.
- “RMI_PDEV_COMMUNICATE returns INCOMPLETE” means that the RmiDevCommExitFlags indicate that device communication is not complete.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



DRAFT

Figure A9.2: PDEV state transitions

A9.2.3.2.1 State transitions for a device which uses SPDM communication

I_{FLKXB} While a PDEV which uses SPDM communication 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.

I_{ZJNMR} While a PDEV which uses SPDM communication is in PDEV_HAS_KEY state, execution of RMI_PDEV_COMMUNICATE causes the RMM to perform secure SPDM session establishment.

Secure SPDM session establishment includes verification of the signature in the SPDM KEY_EXCHANGE response.

Once secure SPDM session establishment is complete, the PDEV moves to PDEV_READY state.

A9.2.3.2.2 State transitions for a device which uses platform communication

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{TSVNW} While a PDEV which uses platform communication is in PDEV_NEW state, execution of RMI_PDEV_COMMUNICATE causes the RMM to establish a communication channel with the device.

Once device identity evidence retrieval is complete, the PDEV moves to PDEV_READY state.

A9.2.3.2.3 State transitions for all devices

I_{WLWSH} While a device transaction is active for the PDEV, 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 either PDEV_READY state or PDEV_ERROR state.

I_{SPTHK} 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.

I_{SRKJH} While a PDEV is in any of the following states, 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_ERROR

A device transaction is initiated for the PDEV, with the operation set to PDEV_OP_STOP.

I_{YZJTQ} While a PDEV has its operation set to PDEV_OP_STOP, 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.

I_{CTRHZ} While a PDEV has its operation set to PDEV_OP_STOP, 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.

I_{VKLQW} 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.

I_{JFRWD} 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
NULL	PDEV_NEW	RMI_PDEV_CREATE
PDEV_NEW	PDEV_NEW	RMI_PDEV_ABORT RMI_PDEV_COMMUNICATE
PDEV_NEW	PDEV_NEEDS_KEY	RMI_PDEV_COMMUNICATE
PDEV_NEW	PDEV_READY	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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

From state	To state	Events
PDEV_HAS_KEY	PDEV_READY	RMI_PDEV_COMMUNICATE
PDEV_NEW	PDEV_STOPPED	RMI_PDEV_STOP
PDEV_NEEDS_KEY	PDEV_STOPPED	RMI_PDEV_STOP
PDEV_HAS_KEY	PDEV_STOPPED	RMI_PDEV_STOP
PDEV_ERROR	PDEV_STOPPED	RMI_PDEV_STOP
PDEV_STOPPED	*NULL*	RMI_PDEV_DESTROY

See also:

- B4.5.16 *RMI_PDEV_ABORT command*
- B4.5.17 *RMI_PDEV_COMMUNICATE command*
- B4.5.18 *RMI_PDEV_CREATE command*
- B4.5.19 *RMI_PDEV_DESTROY command*
- B4.5.21 *RMI_PDEV_SET_PUBKEY command*
- B4.5.22 *RMI_PDEV_STOP command*

See also:

- *PCI Express 6.0 specification* [16]
- A9.5 *Communication between RMM and a device*

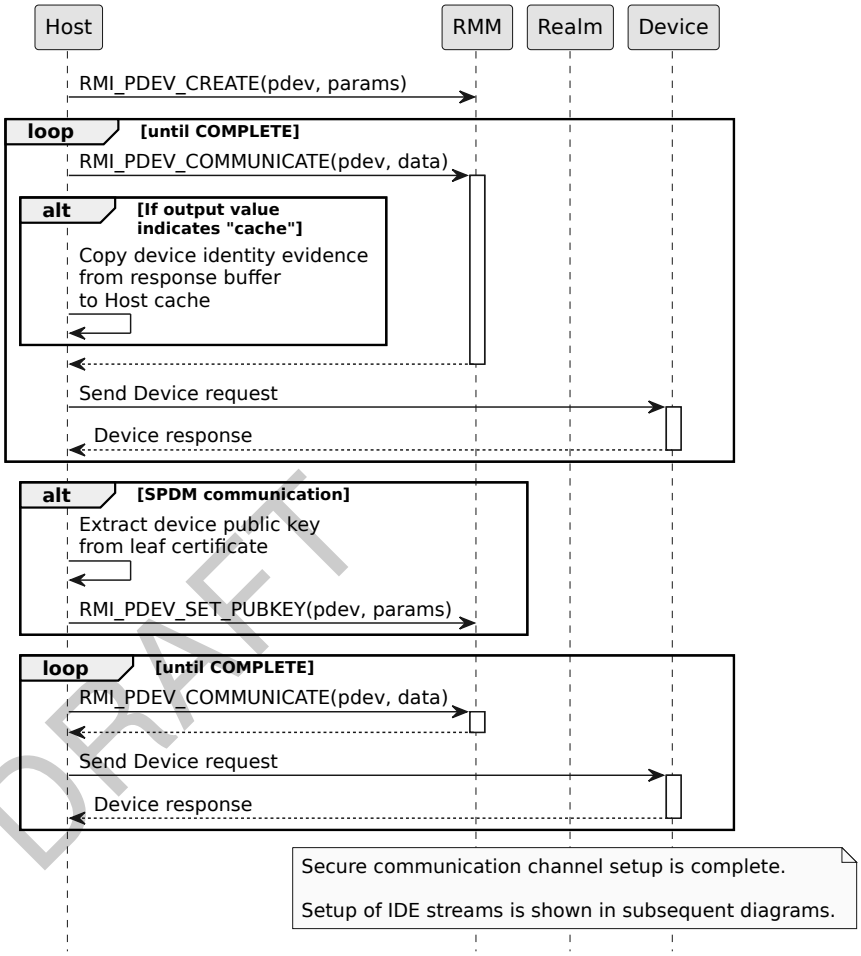
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.2.4 Physical device flows

A9.2.4.1 Physical device setup flow

I_{TPVTS} Setup of a PDEV is illustrated in the following sequence diagram.



DRAFT

Figure A9.3: Setup of a PDEV

I_{PJSQG} Establishment of a Secure SPDM session means that the requester (the RMM) has verified that the responder holds the private key which corresponds to the public key in the device leaf certificate. The identity and trustworthiness of the responder are not evaluated until the Realm receives attestation evidence for the device.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{SPGYF} Mapping of the PDEV setup flow onto SPDM communication with a TDISP PCIe device is illustrated in the following sequence diagram.

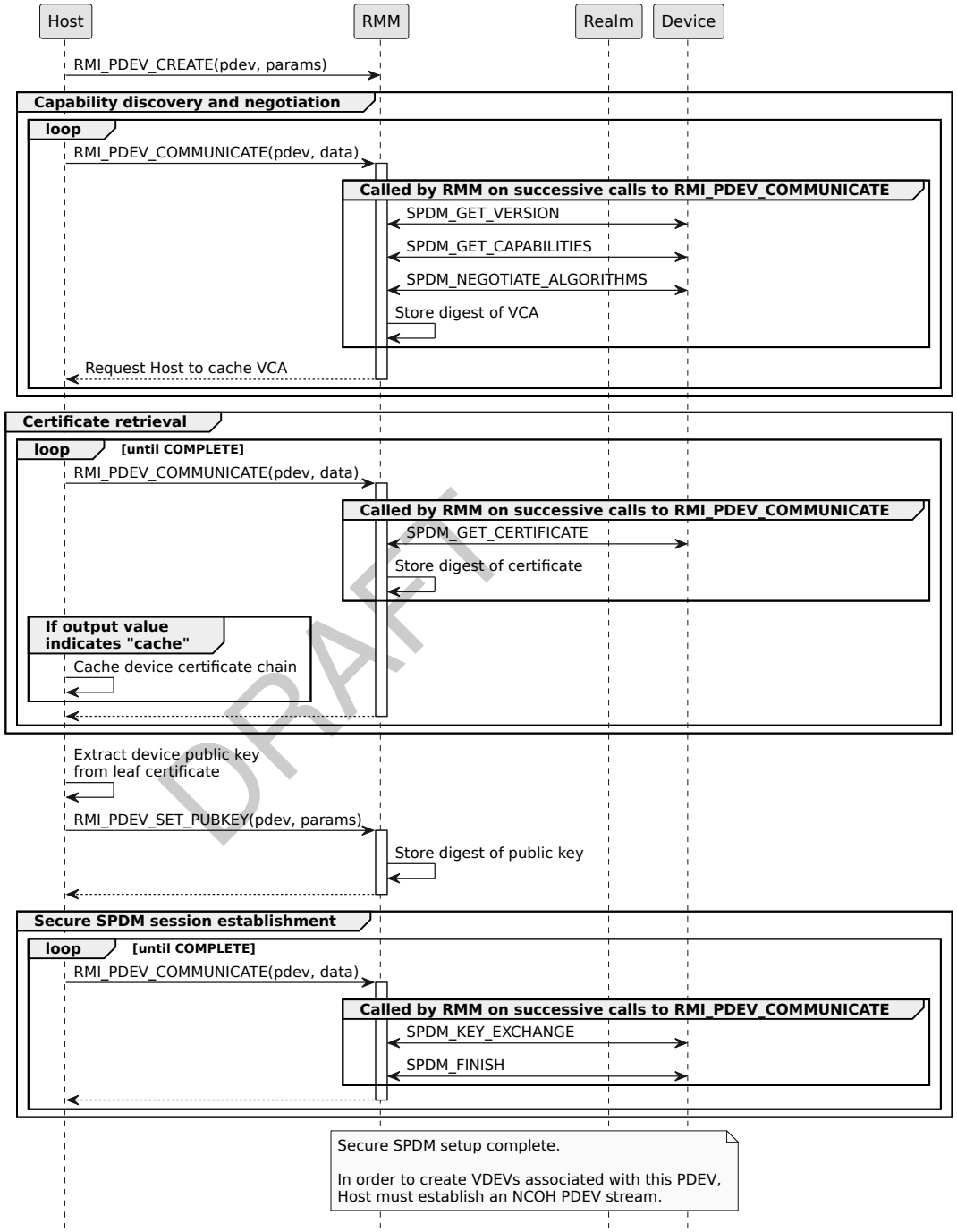


Figure A9.4: Mapping of the PDEV setup flow onto SPDM communication with a TDISP PCIe device

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- *PCI Express 6.0 specification* [16]
- A9.5 *Communication between RMM and a device*
- B4.5.17 *RMI_PDEV_COMMUNICATE command*
- B4.5.18 *RMI_PDEV_CREATE command*
- B4.5.21 *RMI_PDEV_SET_PUBKEY command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.3 Physical device stream object

D_{ZJKLQ} A *Physical Device Stream* (PDEV stream) represents a category of traffic to an endpoint device.

I_{XSYXF} A PDEV stream may represent any of the following:

- Traffic between a Root Port an an off-chip endpoint device
- Traffic between the system and an on-chip endpoint device

D_{HLMRX} The supported categories of PDEV stream are summarized in the following table.

Name	Description
PDEV_STREAM_COH	Coherent traffic between an upstream port and an accelerator endpoint device.
PDEV_STREAM_COH_SYS	Coherent traffic to an endpoint device which is protected by system construction.
PDEV_STREAM_NCOH	Non-coherent traffic between an upstream port and an endpoint device.
PDEV_STREAM_NCOH_SYS	Non-coherent traffic to an endpoint device which is protected by system construction.
PDEV_STREAM_NON_TEE	Non-TEE traffic.

I_{HGDFC} Connection of a PDEV stream is managed by the Host by execution of RMI commands.

I_{VGNJY} Creation of a VDEV is conditional on a sufficient set of PDEV streams having first been established. Satisfaction of this requirement is represented by the return value from the PdevStreamsForVdev() function, which is called by an RMI_VDEV_CREATE pre-condition.

I_{VFFWG} Traffic paths which are not protected by system construction may include switches only if they are pass-through.

See also:

- A9.2 *Physical device object*
- A9.4 *Virtual device object*
- B3.75 *PdevStreamsForVdev function*
- B4.5.24 *RMI_PDEV_STREAM_CONNECT command*
- B4.5.74 *RMI_VDEV_CREATE command*

A9.3.1 Physical device stream attributes

D_{QCFDL} The attributes of a PDEV stream are summarized in the following table.

Name	Type	Description
handle	Bits64	Stream handle
state	RmmPdevStreamState	Stream state
stream_type	RmmPdevStreamType	Stream type
ide_sid	UInt64	IDE stream identifier
num_addr_range	UInt64	Number of device address ranges
addr_range	RmmAddrRange[16]	CPU physical address range used to access the device in the system memory map

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

D_{MQDDS} The meaning and validity of some PDEV stream attributes depends on the stream type. This is summarized in the following table.

A blank cell indicates that the corresponding RmiPdevStreamParams attribute is ignored by RMI_PDEV_STREAM_CONNECT.

stream_type	pdev_1	pdev_2	ide_sid	addr_range	IDE stream type
NON_TEE	Endpoint device	Root Port	Required		PCIe Link IDE stream
NCOH	Endpoint device	Root Port	Required	Required	PCIe Selective IDE stream
COH	Endpoint device	Root Port		Required	CXL.cachemem IDE
NCOH_SYS	Endpoint device			Required	Not Applicable
COH_SYS	Endpoint device			Required	Not Applicable

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{DSZBR} The following table shows the set of PDEV streams which are typically required for each of a set of use cases.

A * indicates that the stream type is optional, depending on the device implementation.

Use case	PDEV stream type				
	NON_TEE	COH	NCOH_SYS	COH_SYS	NCOH
PCIe NON_TEE traffic protection	Y	N	N	N	N
PCIe off-chip	*	N	N	N	Y
PCIe on-chip	N	N	Y	N	N
CXL / CHI off-chip accelerator	N	Y	N	N	*
CXL / CHI on-chip accelerator	N	N	*	Y	N

DRAFT

A9.3.2 Physical device stream invariants

This section lists invariants which are enforced via checks performed by the RMM on execution of RMI_PDEV_STREAM_CONNECT.

R_{PHXKJ} All entries in stream.addr_range have the following properties:

- Not used by any other PDEV stream.
- If the stream type is NCOH or NCOH_SYS, fall within the physical address region(s) of the system memory map which are reserved for non-coherent device memory.
- If the stream type is COH or COH_SYS, fall within the physical address region(s) of the system memory map which are reserved for coherent device memory.

R_{NVLDN} For a stream whose type is NCOH or COH the range `[pdev_1.rid_base, pdev_1.rid_top)` does not overlap any other device in the same PCIe segment.

R_{QGJQB} For a stream whose type is NCOH the `ide_sid` does not match any existing stream in either device.

See also:

- A2.3.2 *Views of physical memory*
- A2.3.4 *Granule tracking region*
- B4.5.24 *RMI_PDEV_STREAM_CONNECT command*

A9.3.3 Physical device stream lifecycle

A9.3.3.1 States

D_{VZFBD} The states of a PDEV stream are listed below.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

State	Description
PDEV_STREAM_DISCONNECTED	Stream is not connected.
PDEV_STREAM_CONNECTING	Stream is connecting.
PDEV_STREAM_CONNECTED	Stream is connected.
PDEV_STREAM_DISCONNECTING	Stream is disconnecting.
PDEV_STREAM_KEY_REFRESHING	Stream keys are being refreshed.
PDEV_STREAM_KEY_PURGING	Stream keys are being purged.

A9.3.3.2 State transitions

I_{VVHZT} Permitted PDEV stream 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
PDEV_STREAM_DISCONNECTED	PDEV_STREAM_CONNECTING	RMI_PDEV_STREAM_CONNECT
PDEV_STREAM_CONNECTING	PDEV_STREAM_CONNECTED	RMI_PDEV_STREAM_COMPLETE
PDEV_STREAM_CONNECTED	PDEV_STREAM_DISCONNECTING	RMI_PDEV_STREAM_DISCONNECT
PDEV_STREAM_DISCONNECTING	PDEV_STREAM_DISCONNECTED	RMI_PDEV_STREAM_COMPLETE
PDEV_STREAM_CONNECTED	PDEV_STREAM_KEY_REFRESHING	RMI_PDEV_STREAM_KEY_REFRESH
PDEV_STREAM_KEY_REFRESHING	PDEV_STREAM_CONNECTED	RMI_PDEV_STREAM_COMPLETE
PDEV_STREAM_CONNECTED	PDEV_STREAM_KEY_PURGING	RMI_PDEV_STREAM_KEY_PURGE
PDEV_STREAM_KEY_PURGING	PDEV_STREAM_CONNECTED	RMI_PDEV_STREAM_COMPLETE

A9.3.3.2.1 Stream connection

I_{WMCXY} Execution of RMI_PDEV_STREAM_CONNECT initiates connection of the stream.

I_{TPGZH} RMI_PDEV_STREAM_CONNECT fails if either PDEV already has the maximum number of streams.

I_{GYXYM} RMI_PDEV_STREAM_CONNECT fails if the endpoint PDEV already has a stream of the specified type.

I_{RSCKV} RMI_PDEV_STREAM_CONNECT fails if the state of either PDEV is not PDEV_READY.

I_{GLYZR} RMI_PDEV_STREAM_CONNECT fails if the operation of either PDEV is not PDEV_OP_NONE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{XSZZV} On successful execution of RMI_PDEV_STREAM_CONNECT, all of the following are true:

- A stream handle is returned. Note that this is an IMPLEMENTATION DEFINED value which identifies the stream, and is distinct from the hardware stream ID.
- The stream state is PDEV_STREAM_CONNECTING.
- For each of the PDEV(s) associated with the stream a device transaction is initiated and the device operation is PDEV_OP_CONNECT.

I_{VZVBJ} Following execution of RMI_PDEV_STREAM_CONNECT, if RMI_PDEV_COMMUNICATE with one PDEV sets the RmiDevCommExitFlags::stream_wait then device communication with the other PDEV must proceed before the the first device communication can proceed.

I_{WRLHB} Following execution of RMI_PDEV_STREAM_CONNECT, when device communication is complete with either PDEV, the device operation of that PDEV is set to PDEV_OP_STREAM_COMPLETE.

I_{DTGGH} Following execution of RMI_PDEV_STREAM_CONNECT, when device communication is complete with both PDEVs, execution of RMI_PDEV_STREAM_COMPLETE completes connection of the stream. The stream state transitions to PDEV_STREAM_CONNECTED.

I_{CNKPG} For details of Root Port IDE key programming which is performed during PDEV stream connection, refer to *Firmware Interfaces for RME (FIRME) specification* [19].

See also:

- *Firmware Interfaces for RME (FIRME) specification* [19]
- A9.5 *Communication between RMM and a device*
- B4.5.23 *RMI_PDEV_STREAM_COMPLETE command*
- B4.5.24 *RMI_PDEV_STREAM_CONNECT command*

A9.3.3.2.2 Stream disconnection

I_{LFDYY} Execution of RMI_PDEV_STREAM_DISCONNECT initiates disconnection of the stream.

I_{DHRSY} Input values to RMI_PDEV_STREAM_DISCONNECT include the stream handle which was returned by RMI_PDEV_STREAM_CONNECT.

I_{MZRXZ} RMI_PDEV_STREAM_DISCONNECT fails if the endpoint PDEV has a non-zero number of VDEVs.

I_{ZJRPJ} On successful execution of RMI_PDEV_STREAM_DISCONNECT, all of the following are true:

- The stream state is PDEV_STREAM_DISCONNECTING.
- For each of the PDEV(s) associated with the stream a device transaction is initiated and the device operation is PDEV_OP_DISCONNECT.

I_{ZDMQX} Following execution of RMI_PDEV_STREAM_DISCONNECT, when device communication is complete with either PDEV, the device operation of that PDEV is set to PDEV_OP_STREAM_COMPLETE.

I_{ZFWFK} Following execution of RMI_PDEV_STREAM_DISCONNECT, when device communication is complete with both PDEVs, execution of RMI_PDEV_STREAM_COMPLETE completes disconnection of the stream. The stream state transitions to PDEV_STREAM_DISCONNECTED.

See also:

- B4.5.23 *RMI_PDEV_STREAM_COMPLETE command*
- B4.5.25 *RMI_PDEV_STREAM_DISCONNECT command*

A9.3.3.2.3 Stream key refresh

I_{CFHHD} Execution of RMI_PDEV_STREAM_KEY_REFRESH initiates key refresh of a stream.

I_{JYRBD} Input values to RMI_PDEV_STREAM_KEY_REFRESH include the stream handle which was returned by RMI_PDEV_STREAM_CONNECT.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{PKWKD} On successful execution of RMI_PDEV_STREAM_KEY_REFRESH, all of the following are true:

- The stream state is PDEV_STREAM_KEY_REFRESHING.
- For each of the PDEV(s) associated with the stream a device transaction is initiated and the device operation is PDEV_OP_KEY_REFRESH.

I_{KWPNV} Following execution of RMI_PDEV_STREAM_KEY_REFRESH, when device communication is complete with either PDEV, the device operation of that PDEV is set to PDEV_OP_STREAM_COMPLETE.

I_{WJQCZ} Following execution of RMI_PDEV_STREAM_KEY_REFRESH, when device communication is complete with both PDEVs, execution of RMI_PDEV_STREAM_COMPLETE completes key refresh of the stream. The stream state transitions to PDEV_STREAM_CONNECTED.

I_{SZNMZ} PDEV stream key refresh may be required in order to complete unlocking of a VDEV. This requirement is discoverable via RMI_FEATURES.

See also:

- A9.4.3 *Virtual device lifecycle*
- B4.5.5 *RMI_FEATURES command*
- B4.5.23 *RMI_PDEV_STREAM_COMPLETE command*
- B4.5.27 *RMI_PDEV_STREAM_KEY_REFRESH command*

A9.3.3.2.4 Stream key purge

I_{PCMTZ} Execution of RMI_PDEV_STREAM_KEY_PURGE initiates purge of inactive keys from the stream.

I_{VTDCS} Input values to RMI_PDEV_STREAM_KEY_PURGE include the stream handle which was returned by RMI_PDEV_STREAM_CONNECT.

I_{MTDQQ} On successful execution of RMI_PDEV_STREAM_KEY_PURGE, all of the following are true:

- The stream state is PDEV_STREAM_KEY_PURGING.
- For each of the PDEV(s) associated with the stream a device transaction is initiated and the device operation is PDEV_OP_KEY_PURGE.

I_{KNZRB} Following execution of RMI_PDEV_STREAM_KEY_PURGE, when device communication is complete with either PDEV, the device operation of that PDEV is set to PDEV_OP_STREAM_COMPLETE.

I_{CSTJR} Following execution of RMI_PDEV_STREAM_KEY_PURGE, when device communication is complete with both PDEVs, execution of RMI_PDEV_STREAM_COMPLETE completes purge of inactive keys from the stream. The stream state transitions to PDEV_STREAM_CONNECTED.

I_{LHSTX} Purging of inactive keys from streams associated with the device may be required in order to complete unlocking of a VDEV. This requirement is discoverable via RMI_FEATURES.

See also:

- A9.4.3 *Virtual device lifecycle*
- B4.5.5 *RMI_FEATURES command*
- B4.5.23 *RMI_PDEV_STREAM_COMPLETE command*
- B4.5.26 *RMI_PDEV_STREAM_KEY_PURGE command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.3.3.3 Physical device stream setup flow

I_{FGSKS} Setup of a PDEV stream between a Root Port and a PCIe off-chip accelerator endpoint device is illustrated in the following sequence diagram.

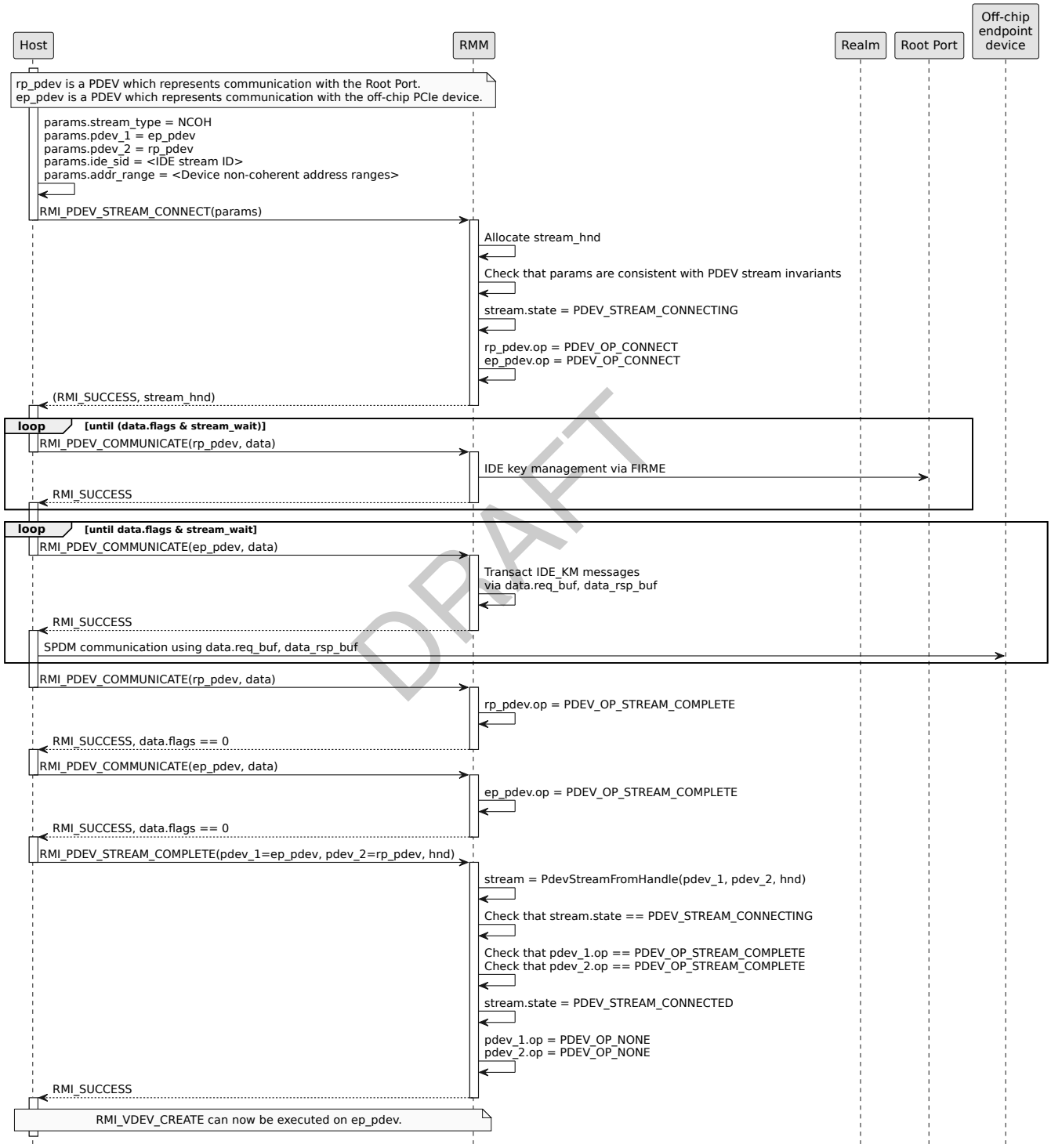


Figure A9.5: Setup of a PDEV stream between a Root Port and a PCIe off-chip accelerator endpoint device

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

IYWBYW Setup of a PDEV stream between a Root Port and a PCIe on-chip accelerator endpoint device is illustrated in the following sequence diagram.

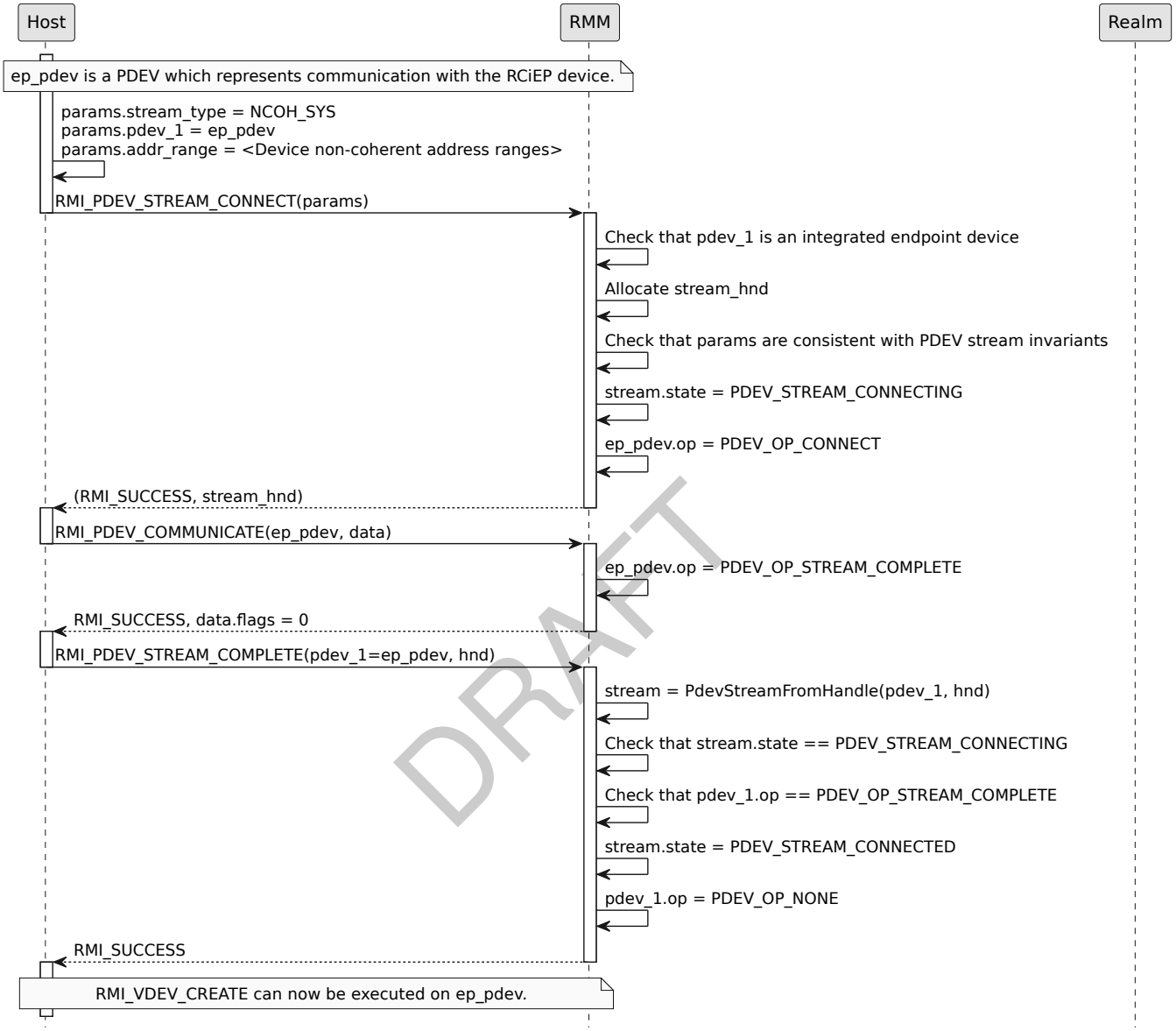


Figure A9.6: Setup of a PDEV stream between a Root Port and a PCIe on-chip accelerator endpoint device

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.4 Virtual device object

D_{LVVJK} A *virtual device* (VDEV) represents the binding between a device function and a Realm.

A9.4.1 Virtual device attributes

D_{NQNVQ} 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
pdev	Address	PA of parent PDEV
realm	Address	PA of RD of Realm which owns this VDEV
vdev_state	RmmVdevState	VDEV lifecycle state
dma_state	RmmVdevDmaState	DMA state
non_ats_plane	UInt64	Index of Plane whose stage 2 permissions are observed by non-ATS requests from the device
op	RmmVdevOperation	Operation performed on this VDEV
comm_state	RmmDevCommState	Device communication state
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.
attest_info	RmmVdevAttestInfo	Attestation information
meas_digest	Bits512	Measurement digest
report_digest	Bits512	Interface report digest
num_addr_range	UInt64	Number of device address ranges
addr_range	RmmAddrRange[8]	CPU physical address range used to access the device in the system memory map These ranges include both coherent and non- coherent device memory.

DRAFT

D_{TJGDD} 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.

I_{HHLFB} 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.

I_{CSJMF} The Host provides the device identifier when executing RMI_VDEV_CREATE.

I_{WKCHQ} The Realm provides the device identifier when executing any RSI_VDEV command.

See also:

- A9.4.4 *Mapping from virtual device ID to VDEV object*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A9.6 *Realm management of an assigned virtual device*
- B4.5.74 *RMI_VDEV_CREATE command*

A9.4.2 Virtual device invariants

This section lists invariants which are enforced via checks performed by the RMM on execution of RMI_VDEV_CREATE.

R_{KXPQP} Execution of RMI_VDEV_CREATE fails if the required Level 2 Stream Table is not present

R_{FQHHK} pdev.num_vdevs is not greater than pdev.max_num_vdevs.

R_{CVHND} vdev.vdev_id is unique among VDEVs assigned to a Realm.

R_{LBQZT} vdev.tdi_id is unique among VDEVs within the same segment.

U_{FSQLB} The RMM can track usage of tdi_id values within a segment either using SW_RESERVED bits in the SMMU Stream Table Entry, or using an IMPLEMENTATION DEFINED data structure stored within the PDEV object.

R_{HJYPM} vdev.tdi_id is within the RID range of the parent PDEV.

R_{ZKBPH} All entries in vdev.addr_range are disjoint.

R_{WMZNP} All entries in vdev.addr_range have the following properties:

- Fall within the union of address ranges for NCOH and COH PDEV streams associated with the parent PDEV.
- Not used by any other VDEV with the same parent PDEV.

See also:

- A9.2.2 *Physical device invariants*
- B4.5.74 *RMI_VDEV_CREATE command*
- B4.5.75 *RMI_VDEV_DESTROY command*

A9.4.3 Virtual device lifecycle

A9.4.3.1 States

D_{MTZBM} The states of a VDEV are listed below.

State	Description
VDEV_NEW	Initial state of the device interface.
VDEV_UNLOCKED	Device interface is unlocked.
VDEV_LOCKED	Device interface is locked.
VDEV_STARTED	Device interface is started.
VDEV_ERROR	Device interface has reported a fatal error.
VDEV_KEY_REFRESH	Waiting for key refresh to be performed on all streams associated with the device interface.
VDEV_KEY_PURGE	Waiting for purge of inactive keys to be performed on all streams associated with the device interface.

X_{BZHJD} The platform may require the Host to perform a key refresh then purge inactive keys from all streams associated with a given device when a VDEV is unlocked. This sequence ensures that any transactions, which were either initiated by the VDEV or which target resources of the VDEV, are completed. This in turn can be used to prevent violations of confidentiality or integrity, which could arise if those transactions were delayed in the IO fabric and later released.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- A9.3.3.2.3 *Stream key refresh*
- A9.3.3.2.4 *Stream key purge*

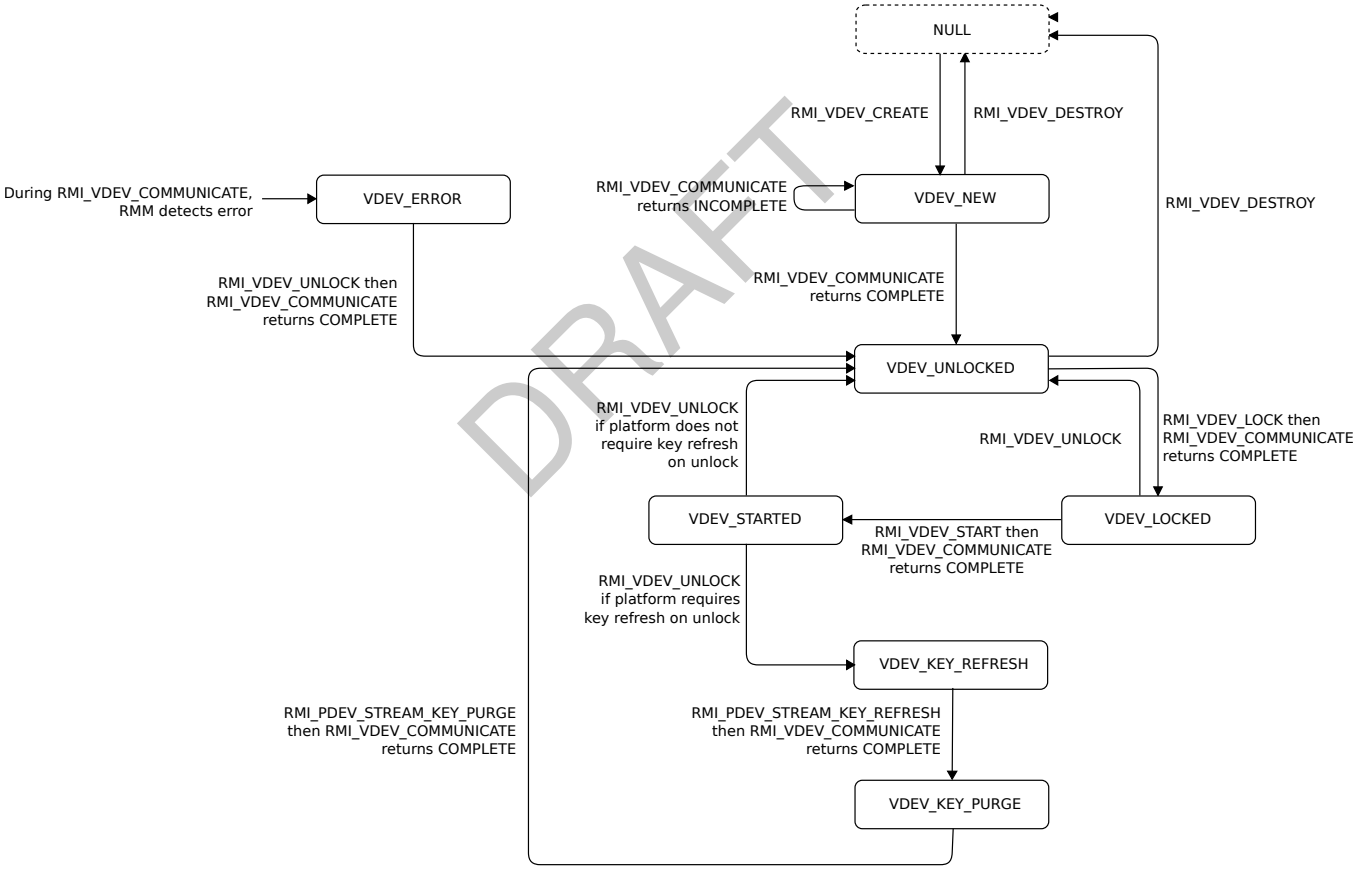
A9.4.3.2 State transitions

I_{RJKJZ} 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.

In this diagram:

- "RMI_VDEV_COMMUNICATE returns COMPLETE" means that the RmiDevCommExitFlags indicate that device communication is complete.
- "RMI_VDEV_COMMUNICATE returns INCOMPLETE" means that the RmiDevCommExitFlags indicate that device communication is not complete.



DRAFT

Figure A9.7: VDEV state transitions

I_{JTVNK} While a VDEV is in VDEV_NEW state, execution of RMI_VDEV_COMMUNICATE causes the RMM to initialize the device.

Once device initialization is complete, the VDEV moves to VDEV_UNLOCKED state.

I_{HJMXW} On execution of RMI_VDEV_UNLOCK, if the state of any Granule within the VDEV's PA ranges is GRAN_DEV then the command fails with RMI_ERROR_GRANULE and returns the PA of the Granule.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{XXRMT} While a VDEV is in VDEV_UNLOCKED state, execution of RMI_VDEV_LOCK causes the RMM to initiate a request to lock the device. The request is sent to the device via subsequent execution of RMI_VDEV_COMMUNICATE.

Once the RMI_VDEV_LOCK request is complete, the VDEV moves to VDEV_LOCKED state.

I_{FMPCY} While a VDEV is in VDEV_LOCKED state, execution of RMI_VDEV_START causes the RMM to initiate a request to start the device. The request is sent to the device via subsequent execution of RMI_VDEV_COMMUNICATE.

Once the RMI_VDEV_START request is complete, the VDEV moves to VDEV_STARTED state.

I_{SSXFB} While a VDEV is in VDEV_LOCKED state, execution of RMI_VDEV_UNLOCK causes the RMM to initiate a request to unlock the device. The request is sent to the device via subsequent execution of RMI_VDEV_COMMUNICATE.

Once the RMI_VDEV_UNLOCK request is complete, the VDEV moves to VDEV_UNLOCKED state.

I_{TNQHL} While a VDEV is in VDEV_STARTED state, execution of RMI_VDEV_UNLOCK causes the RMM to initiate a request to unlock the device. The request is sent to the device via subsequent execution of RMI_VDEV_COMMUNICATE.

Once the RMI_VDEV_UNLOCK request is complete:

- If the platform requires key refresh on VDEV unlock then the VDEV moves to VDEV_KEY_REFRESH state.
- Otherwise the VDEV moves to VDEV_UNLOCKED state.

The requirement for key refresh on VDEV unlock is discoverable via RMI_FEATURES.

I_{CZTPN} While a VDEV is in VDEV_KEY_REFRESH state, the Host is expected to perform a key refresh on all PDEV streams which are associated with the VDEV.

Once the key refreshes are complete, execution of RMI_VDEV_COMMUNICATE causes the VDEV moves to VDEV_KEY_PURGE state.

I_{PHYDK} While a VDEV is in VDEV_KEY_PURGE state, the Host is expected to purge inactive keys from all PDEV streams which are associated with the VDEV.

Once the key purges are complete, execution of RMI_VDEV_COMMUNICATE causes the RMM to initiate a request to unlock the device. The request is sent to the device via subsequent execution of RMI_VDEV_COMMUNICATE.

Once the unlock request is complete, the VDEV moves to VDEV_UNLOCKED state.

I_{YQNQN} 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.

I_{HWZMR} While a VDEV is in VDEV_ERROR state, execution of RMI_VDEV_UNLOCK causes the RMM to initiate a request to unlock the device. The request is sent to the device via subsequent execution of RMI_VDEV_COMMUNICATE.

Once the lock request is complete, the VDEV moves to VDEV_UNLOCKED state.

I_{YTLFV} Execution of RMI_VDEV_UNLOCK fails if the VDEV has associated device memory mappings.

I_{GMJRG} While the state of a VDEV is VDEV_NEW or VDEV_UNLOCKED, the Host can reclaim resources by executing RMI_VDEV_DESTROY.

I_{ZGDVQ} 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

From state	To state	Events
NULL	VDEV_NEW	RMI_VDEV_CREATE
VDEV_NEW	VDEV_NEW	RMI_VDEV_ABORT RMI_VDEV_COMMUNICATE
VDEV_NEW	VDEV_UNLOCKED	RMI_VDEV_COMMUNICATE
VDEV_UNLOCKED	VDEV_LOCKED	RMI_VDEV_COMMUNICATE
VDEV_LOCKED	VDEV_STARTED	RMI_VDEV_COMMUNICATE
VDEV_LOCKED	VDEV_KEY_REFRESH	RMI_VDEV_COMMUNICATE
VDEV_STARTED	VDEV_KEY_REFRESH	RMI_VDEV_COMMUNICATE
VDEV_KEY_REFRESH	VDEV_KEY_PURGE	RMI_VDEV_COMMUNICATE
VDEV_KEY_PURGE	VDEV_UNLOCKED	RMI_VDEV_COMMUNICATE
VDEV_ERROR	VDEV_UNLOCKED	RMI_VDEV_COMMUNICATE
VDEV_NEW	*NULL*	RMI_VDEV_DESTROY
VDEV_UNLOCKED	*NULL*	RMI_VDEV_DESTROY

See also:

- B4.5.71 *RMI_VDEV_ABORT command*
- B4.5.72 *RMI_VDEV_COMMUNICATE command*
- B4.5.74 *RMI_VDEV_CREATE command*
- B4.5.75 *RMI_VDEV_DESTROY command*

See also:

- A9.3.3.2.3 *Stream key refresh*
- A9.3.3.2.4 *Stream key purge*
- B4.5.5 *RMI_FEATURES command*
- B4.5.72 *RMI_VDEV_COMMUNICATE command*
- B4.5.79 *RMI_VDEV_LOCK command*
- B4.5.80 *RMI_VDEV_START command*
- B4.5.81 *RMI_VDEV_UNLOCK command*

A9.4.4 Mapping from virtual device ID to VDEV object

I_{XSMCJ} The RMM does not maintain a mapping from virtual device ID to VDEV object. Conseqeuntly, when a Realm executes an RSI_VDEV command, passing a virtual device ID, the RMM must request the Host to provide a corresponding VDEV object.

I_{GGZRX} The following sequence diagram shows how the virtual device ID 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

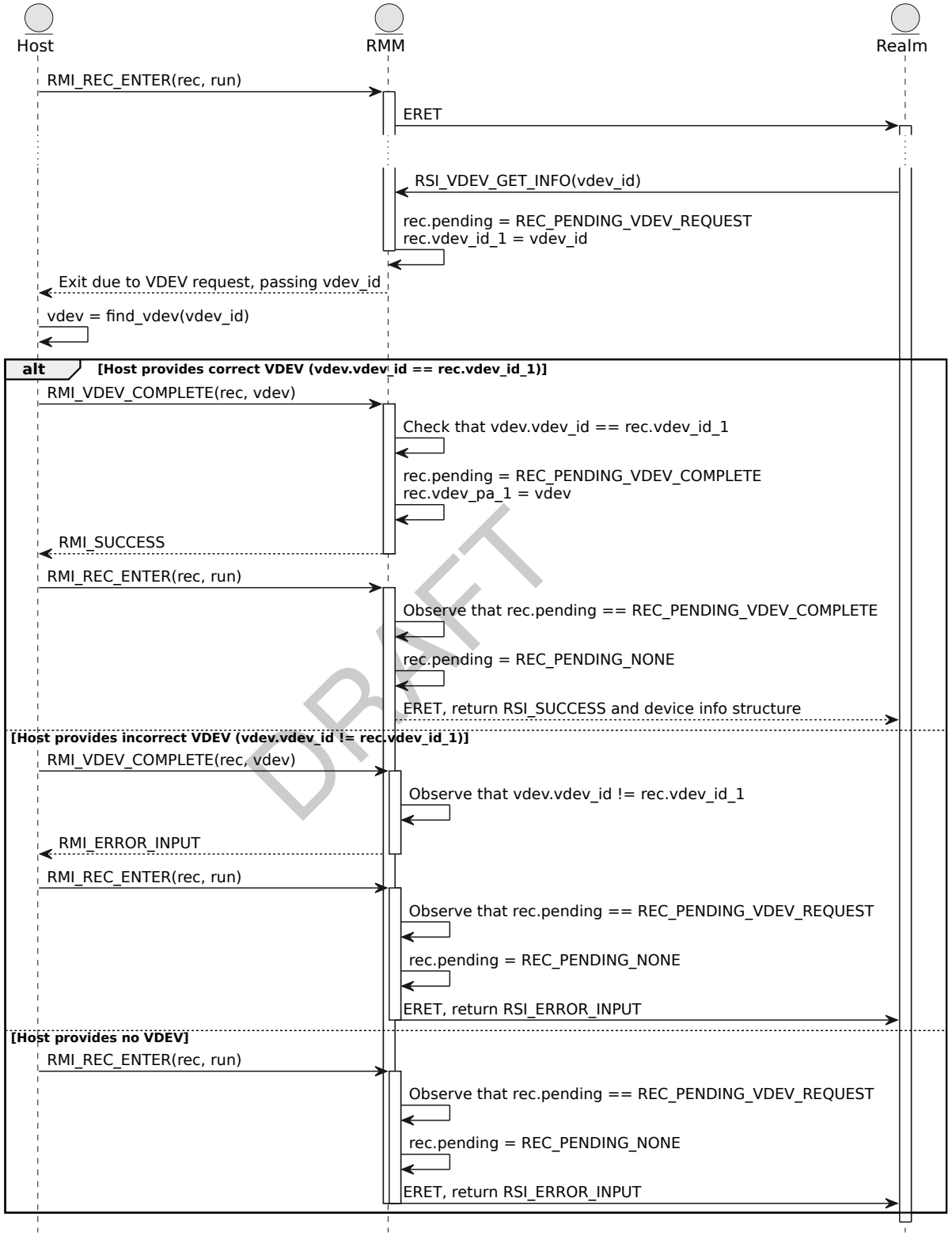


Figure A9.8: Mapping from virtual device ID to VDEV

See also:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A4.3.12 *REC exit due to VDEV request*
- B4.5.73 *RMI_VDEV_COMPLETE command*
- B5.4.19 *RSI_VDEV_GET_INFO command*

A9.4.4.1 Relationship between VDEV state and TDISP TDI state

I_{WRCVK} The lifecycle of a VDEV 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.

R_{WZVWY} On transition of a VDEV to VDEV_LOCKED state, the corresponding TDI transitions to CONFIG_LOCKED state.

R_{KVJKM} On transition of a VDEV to VDEV_STARTED state, the corresponding TDI transitions to RUN state.

R_{ZQHFS} On detection by the RMM that a TDI is in ERROR state, the corresponding VDEV transitions to VDEV_ERROR state.

See also:

- *PCI Express 6.0 specification* [16]

A9.4.4.2 Relationship between VDEV state and SMMU enablement

R_{YDQTT} When the DMA state of a VDEV is VDEV_DMA_ENABLED, the SMMU permits traffic from the device to the Realm's Protected IPA space.

R_{JHFMS} When the DMA state of a VDEV is VDEV_DMA_DISABLED, the SMMU blocks traffic from the device to the Realm's Protected IPA space.

I_{NMPHQ} Execution of RMI_VDEV_UNLOCK caused the DMA state of the VDEV to become VDEV_DMA_DISABLED.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.4.5 Virtual device flows

A9.4.5.1 Virtual device teardown flow

IYSHKG Teardown of a VDEV is illustrated in the following sequence diagram.

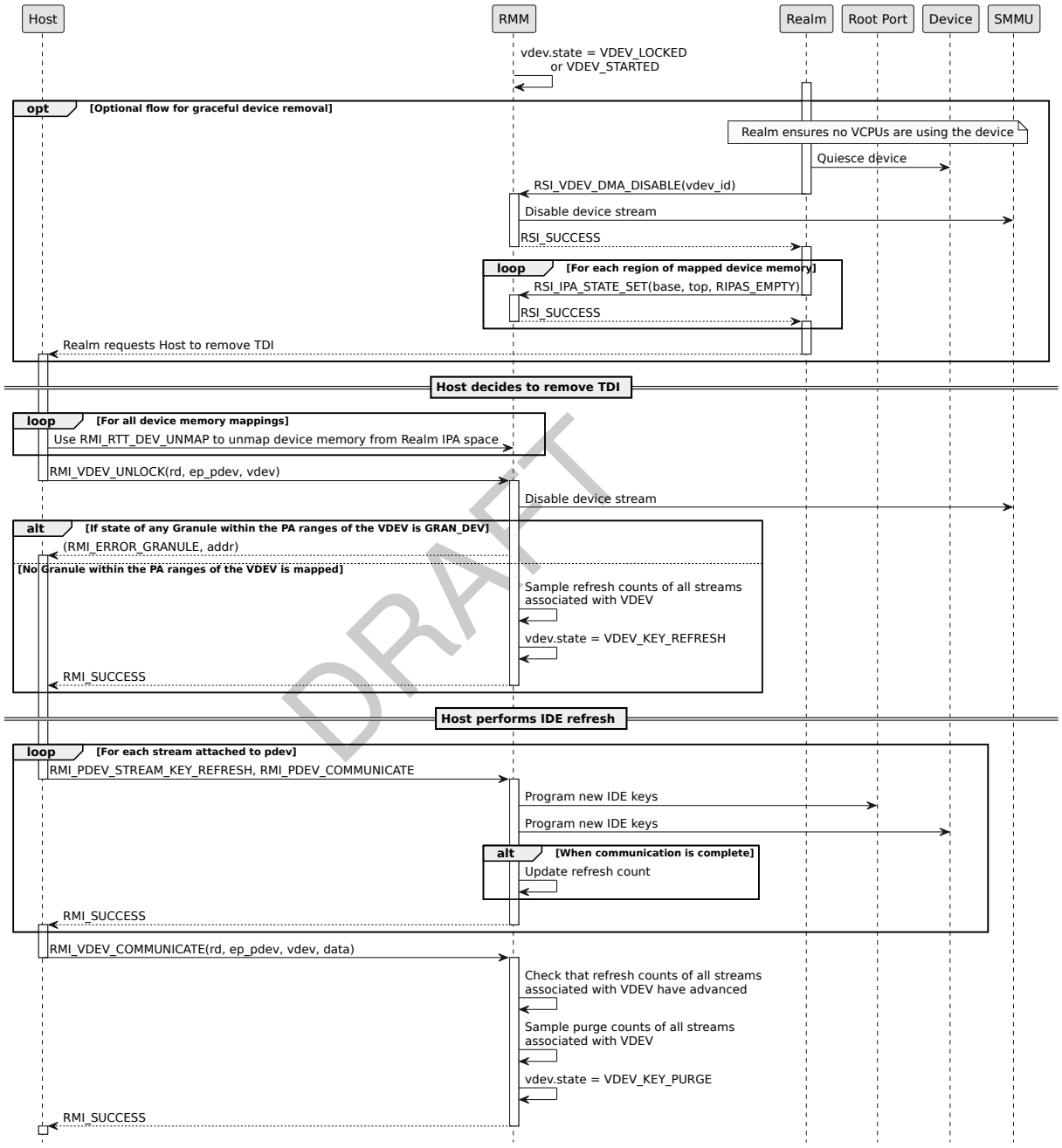


Figure A9.9: VDEV teardown (part 1 of 2)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

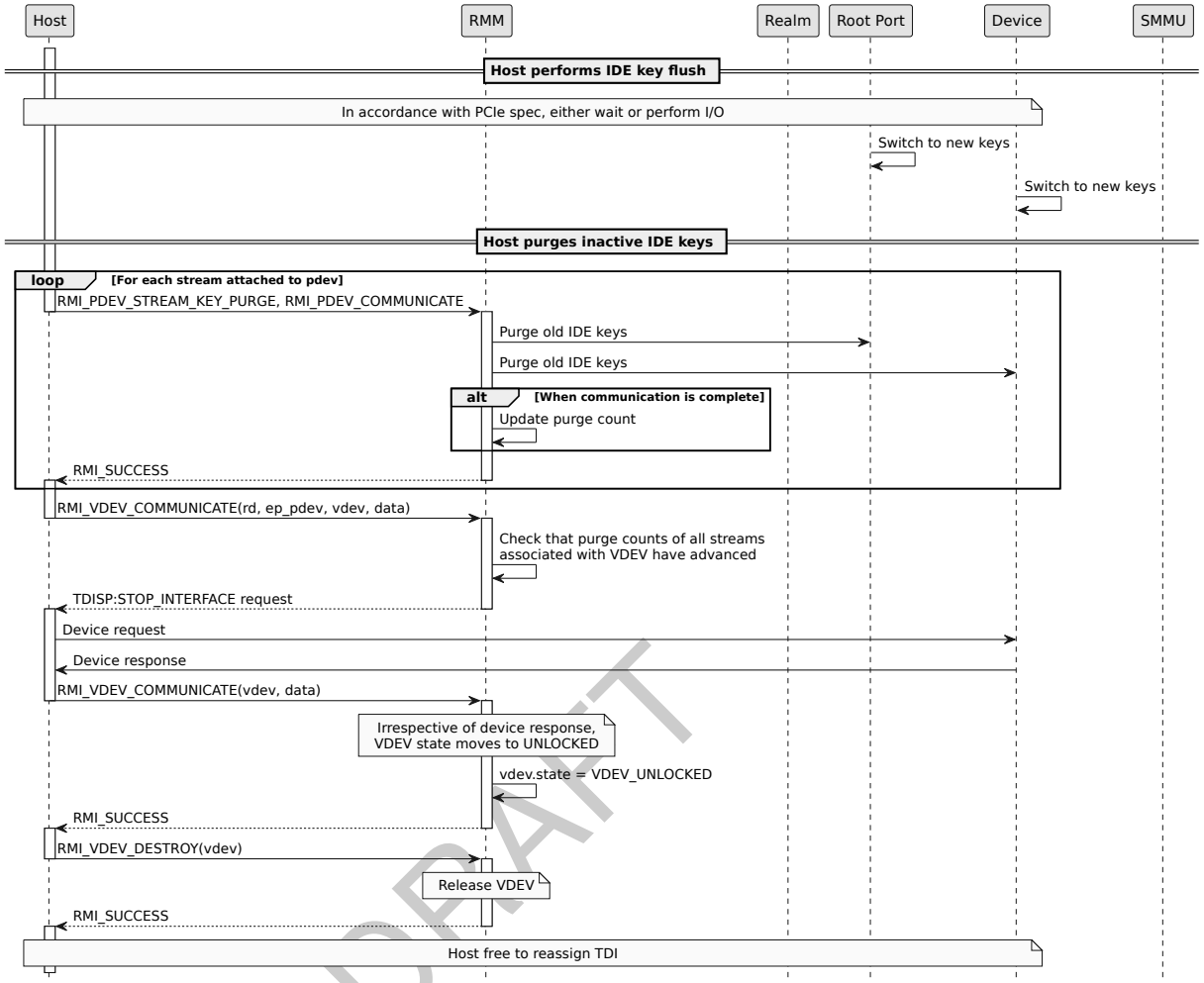


Figure A9.10: VDEV teardown (part 2 of 2)

I_0002 During VDEV teardown, if either the PDEV stream key refresh or the PDEV stream key purge operation fails then:

- The respective count is not updated.
- The Root Port end of the connection has been severed.
- The Host is expected to continue the device communication until the VDEV unlock process is completed.

If the device communication failure was due to the PDEV having transitioned to PDEV_ERROR state then:

- The Root Port end of the connection has been severed.
- RMI_VDEV_UNLOCK causes the VDEV to move to VDEV_UNLOCKED state. RMI_VDEV_DESTROY can subsequently be executed.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I0003 Error handling during teardown of a VDEV is illustrated in the following sequence diagram.

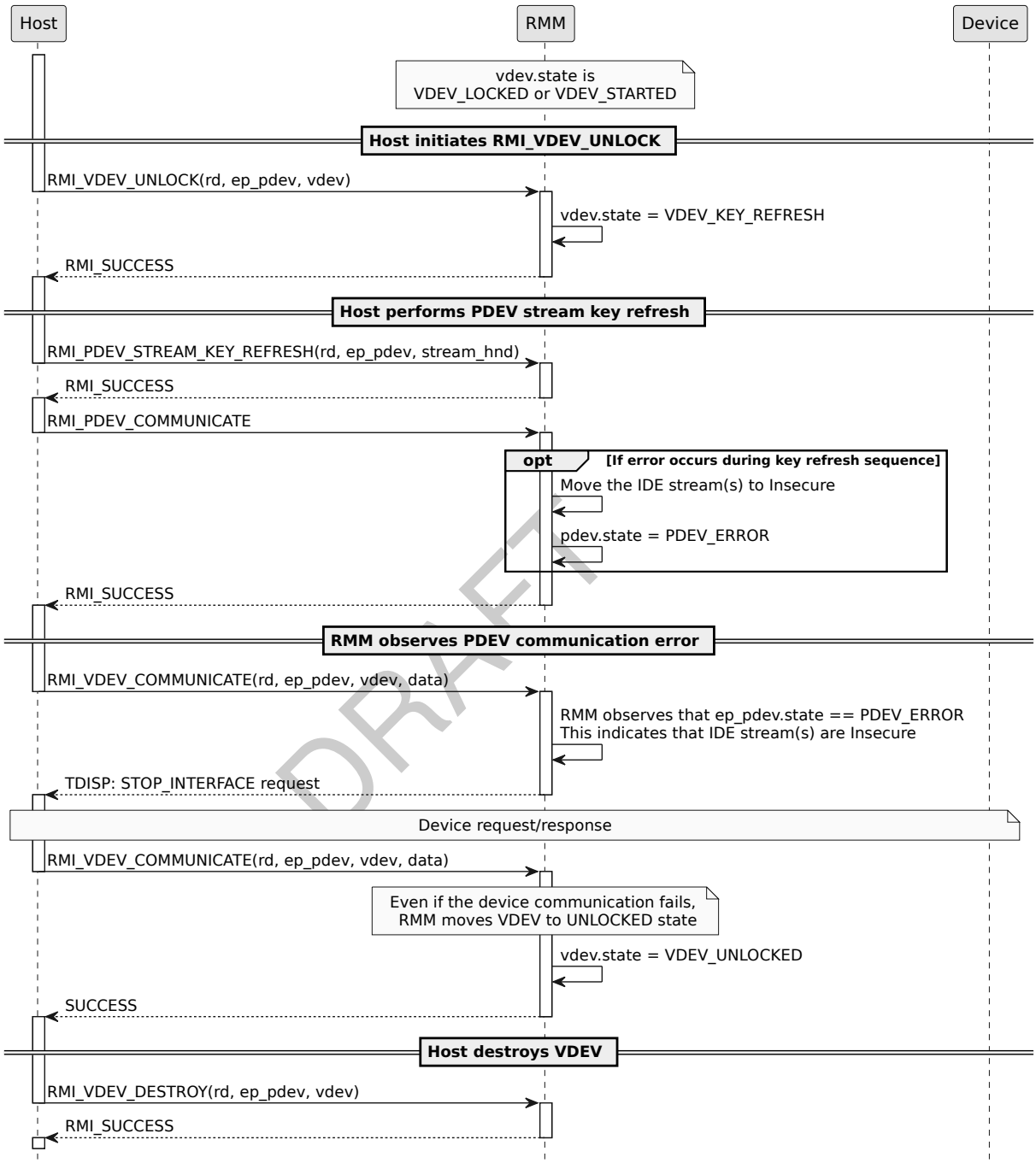


Figure A9.11: VDEV teardown error handling

See also:

- B4.5.62 *RMI_RTT_DEV_UNMAP command*
- B4.5.72 *RMI_VDEV_COMMUNICATE command*
- B4.5.75 *RMI_VDEV_DESTROY command*
- B4.5.81 *RMI_VDEV_UNLOCK command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.5 Communication between RMM and a device

This section describes how the RMM communicates management requests to the device.

See also:

- A9.1.2 *Device properties*

A9.5.1 Device requests and responses

D_{YTBZW} 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.

D_{YWVZH} A *device transaction* is a series of one or more (device request, device response) tuples.

I_{FXWLY} A device transaction is initiated by Host execution of an RMI command.

I_{SQSLN} 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_STOP
  - RMI_PDEV_STREAM_CONNECT
  - RMI_PDEV_STREAM_DISCONNECT
  - RMI_PDEV_STREAM_KEY_REFRESH
- If the device transaction was triggered by one of the following RMI commands then the device transaction is associated with the VDEV.
  - RMI_VDEV_GET_INTERFACE_REPORT
  - RMI_VDEV_GET_MEASUREMENTS
  - RMI_VDEV_LOCK
  - RMI_VDEV_START
  - RMI_VDEV_UNLOCK

DRAFT

I_{RQRQF} A Realm is expected to request the Host to initiate device transactions (for both state changes and for requesting measurements and interface report) via an RSI_HOST_CALL interface. For details, refer to *Realm Host Interface specification* [20].

I_{FHBLL} A PDEV is associated with at most one device transaction at a time.

I_{LPSZW} A VDEV is associated with at most one device transaction at a time.

D_{NLVBV} 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.
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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{PSRPH} Device communication state transitions are shown in the following figure. Each arc is labeled with the events which can cause the corresponding state transition.

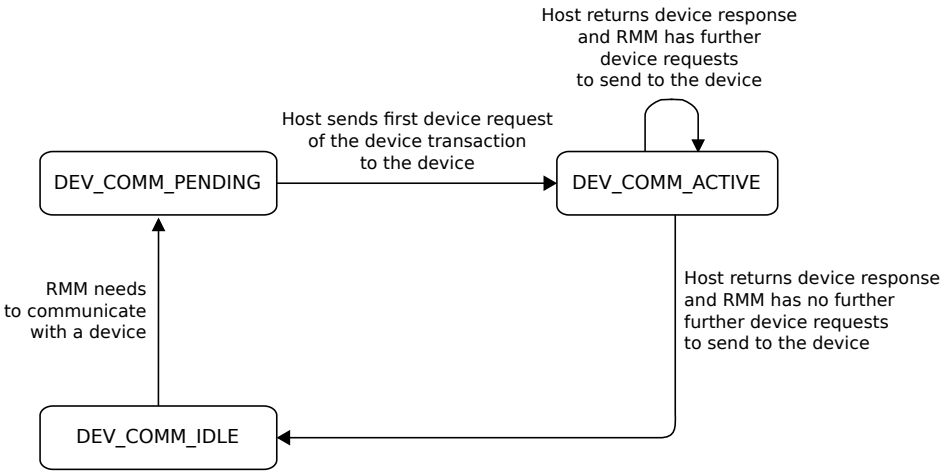


Figure A9.12: Device communication state transitions

See also:

- *Secured Messages using SPDM Specification version 1.1.0* [17]
- B4.6.17 *RmiDevCommExitFlags type*

I_{BDHSP} While the device communication state of a PDEV is either DEV_COMM_PENDING or DEV_COMM_ACTIVE, the Host can either:

- Transfer device requests and device responses by executing RMI_PDEV_COMMUNICATE.
- Abort the device transaction by executing RMI_PDEV_ABORT.

I_{MXRWH} While the device communication state of a VDEV is either DEV_COMM_PENDING or DEV_COMM_ACTIVE, the Host can either:

- Transfer device requests and device responses by executing RMI_VDEV_COMMUNICATE.
- Abort the device transaction by executing RMI_VDEV_ABORT.

I_{GCYRR} If a physical device is temporarily unable to service a request for an IMPLEMENTATION DEFINED reason then RMI_PDEV_COMMUNICATE or RMI_VDEV_COMMUNICATE may return RMI_BUSY.

S_{SRHCJ} When the Host receives RMI_BUSY from RMI_VDEV_COMMUNICATE, it can choose to either:

- Propagate the busy status to the requesting Realm via RHI.
- Hold onto the Realm's request, and retry the RMI command later. While holding on to the request, the Host should indicate via RHI that the request is incomplete.

See also:

- *Realm Host Interface specification* [20]
- B4.5.17 *RMI_PDEV_COMMUNICATE command*
- B4.5.72 *RMI_VDEV_COMMUNICATE command*

A9.5.2 Device communication data structures

A9.5.2.1 Device communication exit data structure

D_{ZJTGD} An *RmiDevCommExit object* is a data structure which is passed from the RMM to the Host during a device transaction.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{CFHJY} 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::rsp_cache flag.
- Whether there is data in the device request buffer which the Host is requested to cache. This is indicated by the RmiDevCommExitFlags::req_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::req_send flag.
- Whether the Host is requested to delay sending a device request. This is indicated by the RmiDevCommExit::req_delay value.
- Whether the RMM is waiting for a response from the device. This is indicated by the RmiDevCommExitFlags::rsp_wait flag.
- The maximum time which the Host should wait for a response from the device. This is indicated by the RmiDevCommExit::req_timeout value. The time to wait is measured from the most recent object which had the RmiDevCommExitFlags::rsp_reset flag set to true.
- Whether the device transaction contains more than one (device request, device response) tuple. This is indicated by the RmiDevCommExitFlags::multi flag.
- Whether the device transaction is waiting for a transaction on another device, to which this device is connected via a PDEV stream, to proceed. This is indicated by the RmiDevCommExitFlags::stream_wait flag.

D_{MDBXD} *Device communication is complete* means that none of the following flags are set:

- RmiDevCommExitFlags::req_send
- RmiDevCommExitFlags::req_cache
- RmiDevCommExitFlags::rsp_cache
- RmiDevCommExitFlags::rsp_reset
- RmiDevCommExitFlags::rsp_wait
- RmiDevCommExitFlags::stream_wait

I_{XZSST} RmiDevCommExitFlags::req_send and RmiDevCommExitFlags::rsp_wait are never set together.

I_{RRCLW} RmiDevCommExitFlags::multi is only set when RmiDevCommExitFlags::req_send is set.

U_{YYVZK} A device which uses SPDM communication is permitted to respond with ResponseNotReady and a token and timeout value (RDTExponent and RDTM). In this case, the RMM is expected to do the following:

- Construct a RespondIfReady request
- Set RmiDevCommExitFlags::req_delay according to the RDTExponent and RDTM values

As a result, the Host is expected to wait for the specified amount of time, before it sends the RespondIfReady request to the device.

U_{SPLZZ} When initiating a new SPDM request, the RMM is expected to set the RmiDevCommExitFlags::rsp_reset flag. On subsequent RMI_xDEV_COMMUNICATE calls while the SPDM request is still outstanding, the RMM is expected to clear the RmiDevCommExitFlags::rsp_reset flag.

As a result, the Host is expected to measure the time taken by the SPDM request from the first call in the sequence, when the RmiDevCommExitFlags::rsp_reset flag was set.

I_{RZMWC} During communication between the RMM and a device which uses SPDM:

- RmiDevCommExitFlags::req_send indicates that the Host is requested to send a device request to the device.
- RmiDevCommExitFlags::rsp_wait indicates that the Host is requested to return a device response to the RMM.

I_{SCJZM} During communication between the RMM and a device which does not use SPDM:

- RmiDevCommExitFlags::rsp_wait indicates that the Host is requested to either:
  - Notify the RMM when a device response is available, or
  - Periodically ask the RMM to check whether a device response has been received.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{XFBZH} On return from RMI_PDEV_COMMUNICATE or RMI_VDEV_COMMUNICATE, RmiDevCommExit::req_len is guaranteed to be not greater than the RMI Granule size.

R_{MKMQJ} On return from RMI_PDEV_COMMUNICATE or RMI_VDEV_COMMUNICATE, RmiDevCommExit::req_cache_offset + RmiDevCommExit::req_cache_len is guaranteed to be not greater than the RMI Granule size.

R_{HCNNN} On return from RMI_PDEV_COMMUNICATE or RMI_VDEV_COMMUNICATE, RmiDevCommExit::rsp_cache_offset + RmiDevCommExit::rsp_cache_len is guaranteed to be not greater than the RMI Granule size.

D_{VYFKM} 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
req_cache_offset	0x8	UInt64	If flags.req_cache is true, offset in the device request buffer to the start of data to be cached, in bytes
req_cache_len	0x10	UInt64	If flags.req_cache is true, amount of device request data to be cached, in bytes
rsp_cache_offset	0x18	UInt64	If flags.rsp_cache is true, offset in the device response buffer to the start of data to be cached, in bytes
rsp_cache_len	0x20	UInt64	If flags.rsp_cache is true, amount of device response data to be cached, in bytes
cache_object_id	0x28	RmiDevCommObject	If flags.req_cache is true and / or flags.rsp_cache is true, identifier for the object to be cached
protocol	0x30	RmiDevCommProtocol	If flags.req_send is true, protocol to use
req_delay	0x38	UInt64	If flags.req_send is true, amount of time to wait before sending the request, in microseconds.
req_len	0x40	UInt64	If flags.req_send is true, amount of valid data in request buffer in bytes
rsp_timeout	0x48	UInt64	Amount of time to wait (measured from the most recent exit which had flags.rsp_reset = true) for device response in microseconds.

DRAFT

See also:

- A9.3 *Physical device stream object*
- B4.5.17 *RMI_PDEV_COMMUNICATE command*
- B4.5.72 *RMI_VDEV_COMMUNICATE command*
- B4.6.16 *RmiDevCommExit type*
- B4.6.17 *RmiDevCommExitFlags type*

A9.5.2.2 Device communication enter data structure

D_{PPHJZ} An *RmiDevCommEnter object* is a data structure which is passed from the Host to the RMM during a device transaction.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{WXGVT} 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.

R_{NWZWL} If RmiDevCommEnter::rsp_len is greater than the RMI Granule size then RMI_PDEV_COMMUNICATE or RMI_VDEV_COMMUNICATE returns RMI_ERROR_INPUT.

D_{BZGJF} 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
rsp_addr	0x10	Address	Address of response buffer
rsp_len	0x18	UInt64	Amount of valid data in response buffer in bytes

See also:

- B4.5.17 *RMI_PDEV_COMMUNICATE command*
- B4.5.72 *RMI_VDEV_COMMUNICATE command*
- B4.6.15 *RmiDevCommEnter type*

DRAFT

A9.5.3 Host-side device communication flow

I_{ZVQLX} 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.

I_{NCQHB} 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.

I_{TJCNW} The RMI_PDEV_COMMUNICATE and RMI_VDEV_COMMUNICATE commands have identical programming models. Hereafter, they are referred to collectively as “device communication commands”.

R_{VZMZN} For a given physical device, at most one device transaction can be active.

I_{KFKHW} The RMI_PDEV_ABORT command or RMI_VDEV_ABORT command is used to abort an DEV_COMM_ACTIVE device transaction.

I_{FNCJS} 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.

A9.5.3.1 Communication flow for devices which use SPDM

I_{NZLCF} The overall flow for communication between the RMM and a device which uses SPDM is as follows:

1. The Host executes an RMI command which causes a device transaction, associated with a specified PDEV or VDEV, to become pending.

   The output values of the command include an indication of 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

4. The Host sends the device request to the device.

   Details of how this is performed are out of scope of this specification. For a 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 is performed 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.

I_{RSPTF} 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.

I_{QDBZK} 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.

U_{BMMVH} The SPDM session which is established between the RMM and a device has the following characteristics:

- SPDM heartbeat is not supported
- SPDM key update is not supported

U_{KPPFB} 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.

I_{NLNKJ} The following sequence illustrates communication with a device which uses SPDM, taking RMI_VDEV_LOCK as the example which initiates the communication.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

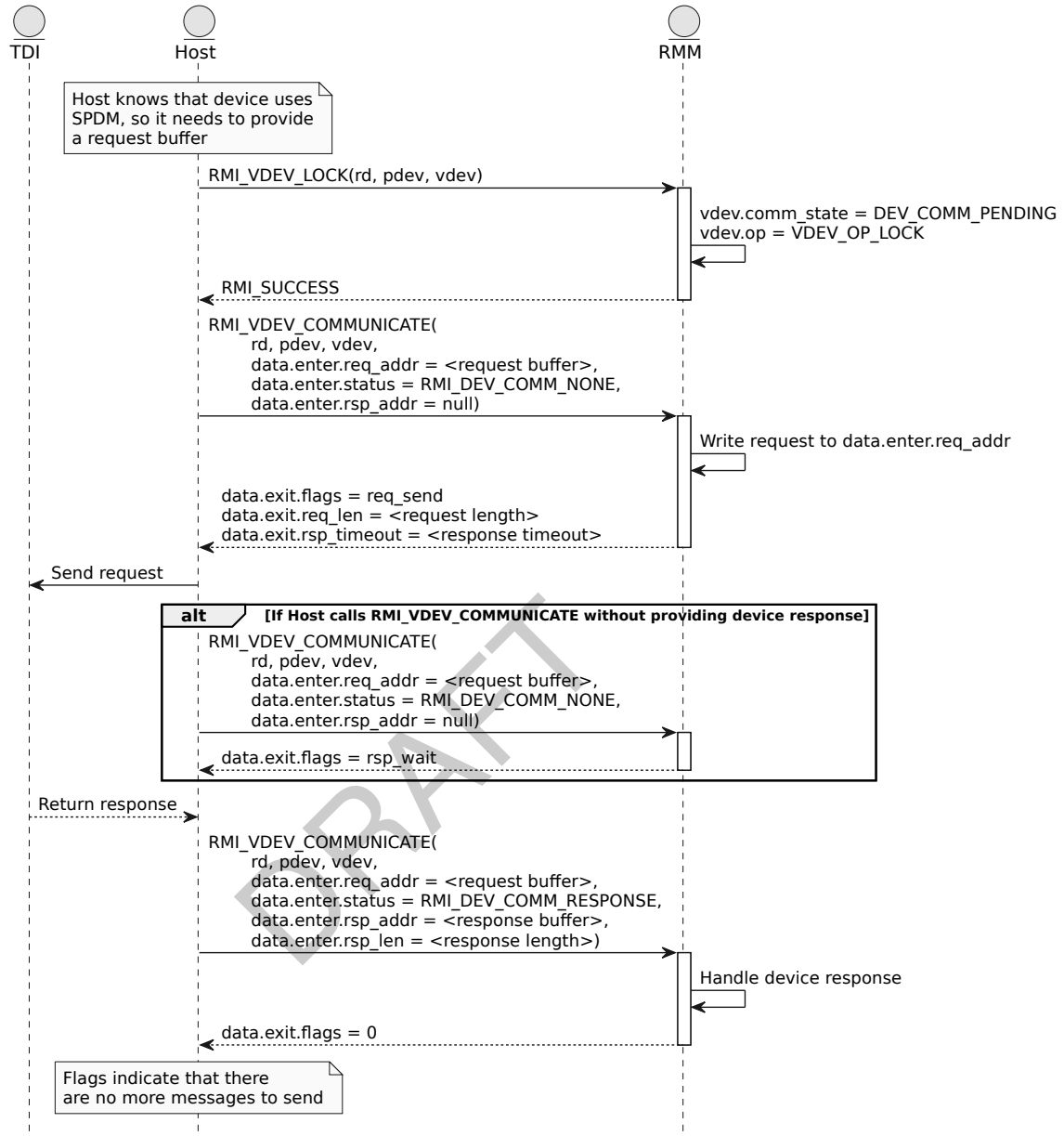


Figure A9.13: Communication with a device which uses SPDM

U_{TMZQY} The implementation should set the SPDM DataTransferSize to a value which allows the largest possible secure SPDM payload to fit within an RMI Granule.

See also:

- *PCI Express 6.0 specification* [16]
- B4.5.17 *RMI_PDEV_COMMUNICATE command*
- B4.5.72 *RMI_VDEV_COMMUNICATE command*

A9.5.3.2 Communication flow for devices which do not use SPDM

I_{XHVYW} The overall flow for communication between the RMM and a device which does not use SPDM is as follows:

1. The Host executes an RMI command which causes a device transaction, associated with a specified PDEV or VDEV, to become pending.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The output values of the command include an indication of whether the pending transaction will contain more than one (device request, device response) tuple.

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.

I_{LFRNJ} When the RMM sends a device request via an IMPLEMENTATION DEFINED channel, the "req_send" flag is clear and the "req_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.

I_{NLBWX} The following sequence illustrates communication with a device which does not use SPDM, taking RMI_VDEV_LOCK as the example which initiates the communication.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

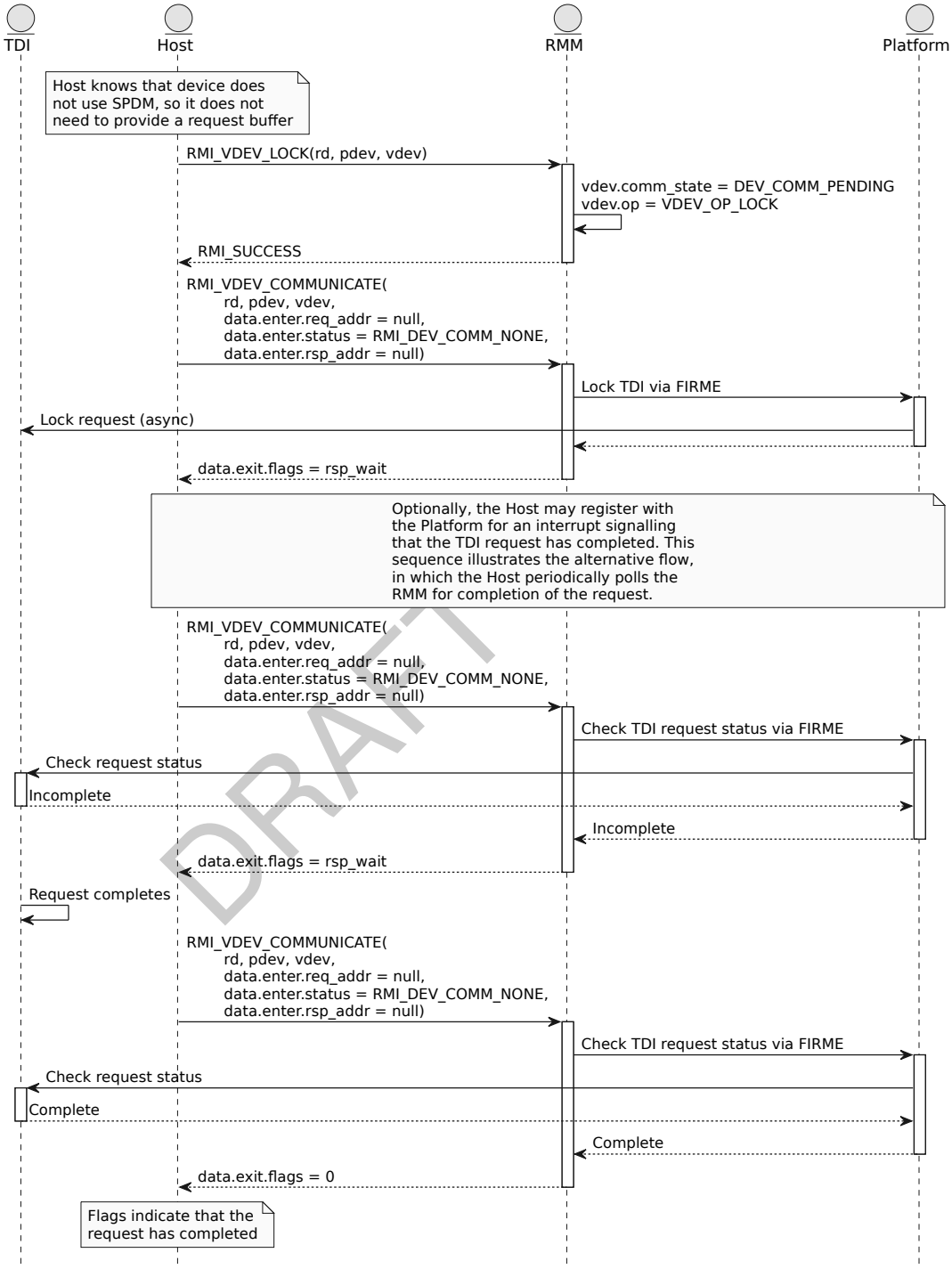


Figure A9.14: Communication with a device which does not use SPDM

I_KQKBY For details of communication with the device via the platform, refer to *Firmware Interfaces for RME (FIRME) specification* [19].

A9.5.4 Host caching of device attestation evidence

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{NWBPX} On execution of a device communication command, the RMM can indicate to the Host that the Host should cache data from the request buffer and / or response buffer, for later retrieval by the Realm.

I_{HBYJL} If RmiDevCommExitFlags::req_cache is set then the Host should cache data from the request buffer, with the extent identified by the RmiDevCommExit::req_cache_offset and RmiDevCommExit::req_cache_len fields.

I_{MPJXV} If RmiDevCommExitFlags::rsp_cache is set then the Host should cache data from the response buffer, with the extent identified by the RmiDevCommExit::rsp_cache_offset and RmiDevCommExit::rsp_cache_len fields.

I_{MSHSD} The identity of the data which the Host is requested to cache is identified by the RmiDevCommExit::cache_object_id field.

I_{SMMYY} If the device transaction was triggered while the PDEV state was PDEV_NEW then the RMM may indicate that the Host should cache the following objects:

- A protocol data object, indicated by RmiDevCommExit::cache_object_id == RMI_DEV_PROTOCOL_DATA.
- A device identity evidence, indicated by RmiDevCommExit::cache_object_id == RMI_DEV_IDENTITY.

I_{QTSCD} If the device transaction was triggered by RMI_VDEV_GET_MEASUREMENTS then the RMM may indicate that the Host should cache device measurement data, indicated by RmiDevCommExit::cache_object_id == RMI_DEV_MEASUREMENTS.

For a device which uses SPDM communication, this data consists of the SPDM GET_MEASUREMENTS request and the corresponding MEASUREMENTS response message, stored in the same sequence used to construct the SPDM measurement transcript. Note that this data does not include the VCA exchange.

For a device which uses platform communication, the format of this data is IMPLEMENTATION DEFINED.

I_{TYCWV} If the device transaction was triggered by RMI_VDEV_GET_INTERFACE_REPORT then the RMM may indicate that the Host should cache a device interface report, indicated by RmiDevCommExit::cache_object_id == RMI_DEV_INTERFACE_REPORT.

See also:

- A9.6.1 *Realm retrieval of device attestation evidence*
- B4.5.18 *RMI_PDEV_CREATE command*
- B4.5.76 *RMI_VDEV_GET_INTERFACE_REPORT command*
- B4.5.77 *RMI_VDEV_GET_MEASUREMENTS command*
- B4.6.16 *RmiDevCommExit type*

A9.5.5 Device measurement retrieval

A9.5.5.1 Device measurement retrieval overview

I_{PKVPT} Device measurement retrieval is initiated by execution of RMI_VDEV_GET_MEASUREMENTS.

I_{RJNBG} Following initiation of device measurement retrieval, the Host is expected to call RMI_VDEV_COMMUNICATE in order to exchange device requests and device responses with the RMM.

I_{GLMYC} During all variants of the device measurement retrieval flow, on return from RMI_VDEV_COMMUNICATE the RMM may request the Host to cache measurement data, for later retrieval by the Realm via RHI.

During a device measurement retrieval flow, the Host is expected to concatenate this data into a single “cached measurement buffer”.

The RMM stores a digest of the measurement data which it requests the Host to cache, which the Realm can request via RSI_VDEV_GET_INFO.

I_{HPBJC} Device measurement retrieval is permitted regardless of the state of the VDEV. The Realm can use the RsiVdevInfo::lock_nonce and RsiVdevInfo::meas_nonce fields to check measurement freshness.

See also:

- A9.5.3 *Host-side device communication flow*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.72 *RMI_VDEV_COMMUNICATE command*
- B4.5.77 *RMI_VDEV_GET_MEASUREMENTS command*
- B5.4.19 *RSI_VDEV_GET_INFO command*

A9.5.5.2 Retrieval of device measurements from a device which uses SPDM

R_{SCKYM} During retrieval of device measurements from a device which uses SPDM, the data which the RMM requests the Host to cache is the SPDM measurement request and response messages.

R_{LXDFR} During retrieval of device measurements from a device which uses SPDM, if the device supports signed measurements then the SPDM_MEASUREMENTS response includes a device-generated signature over the SPDM measurement transcript, as specified in the SPDM specification; see *Security Protocol and Data Model (SPDM)* [21].

I_{BBVNW} The following sequence illustrates retrieval of device measurements from a device which uses SPDM.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

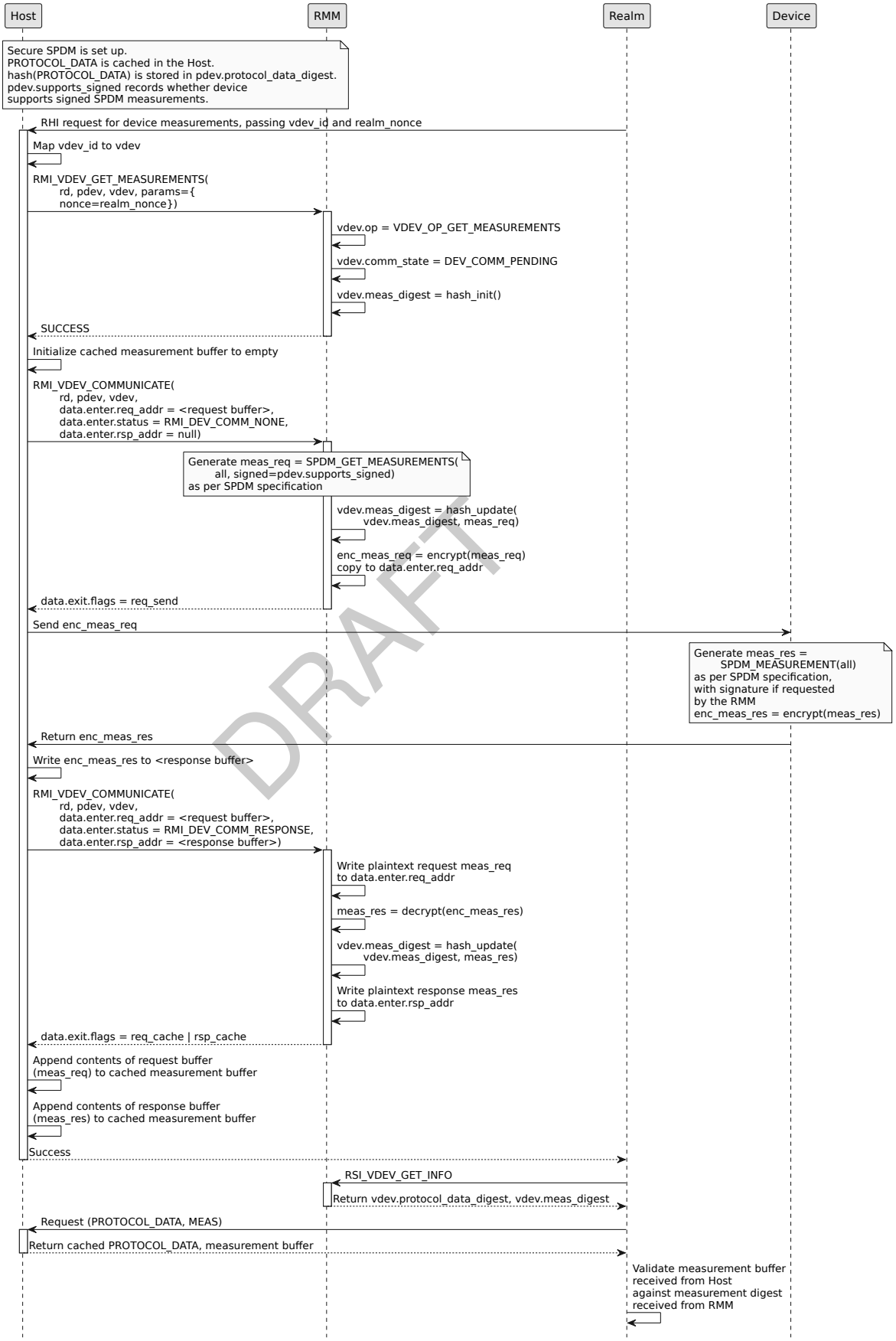
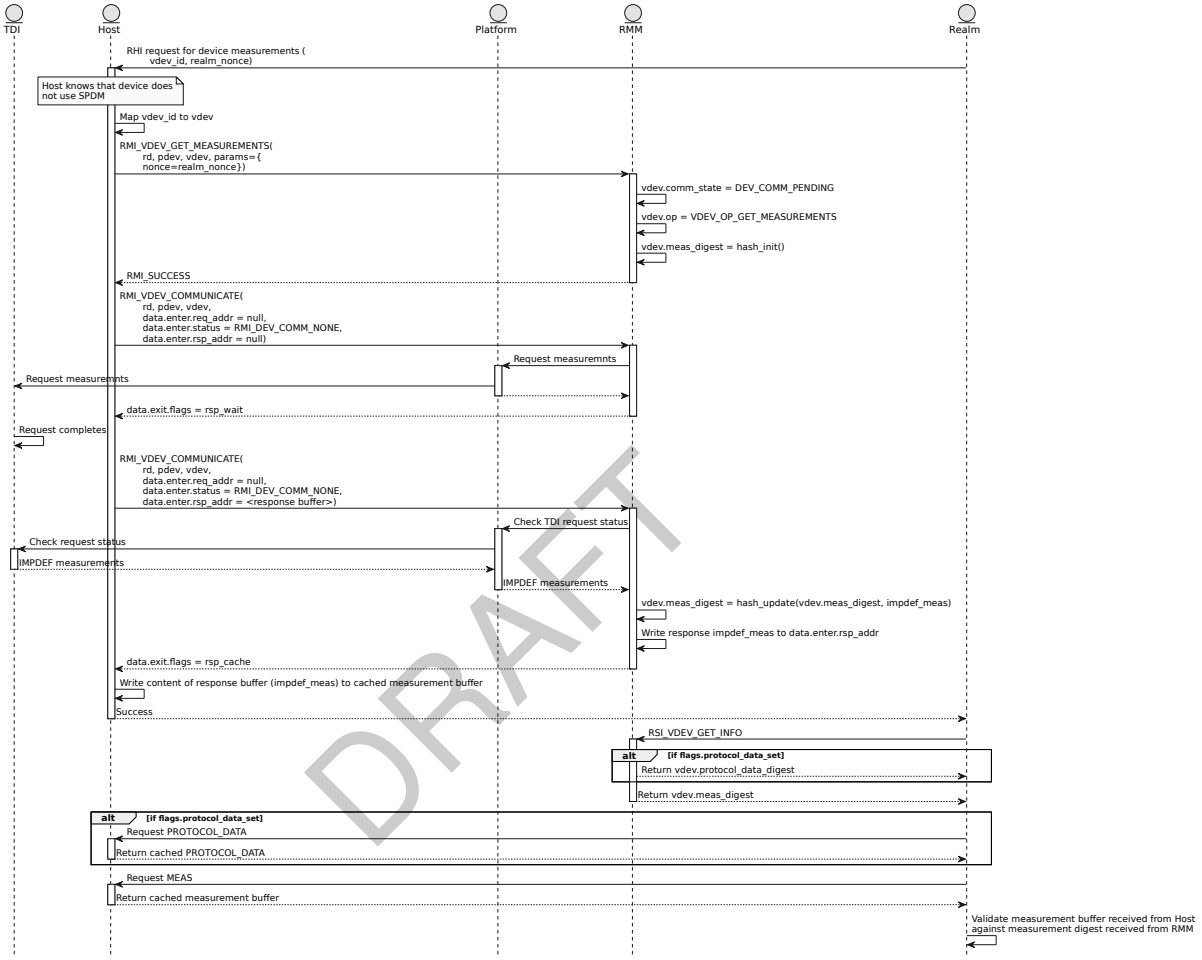


Figure A9.15: Retrieval of measurements from a device which uses SPDM

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.5.5.3 Retrieval of device measurements from a device which does not use SPDM

IZHJZK For a device which does not use SPDM, the flags returned from RMI_VDEV_COMMUNICATE have “rsp_wait” set and “req_send” clear. This indicates that the Host should either poll for completion of the request by calling RMI_VDEV_COMMUNICATE, or that the Host will receive an IMPLEMENTATION DEFINED notification when the request has been completed.



See also:

- A9.5.3 *Host-side device communication flow*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.6 Realm management of an assigned virtual device

This section describes interaction between a Realm and the RMM to manage an assigned virtual device.

I_{GXJLB} 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.

See also:

- A9.4 *Virtual device object*

A9.6.1 Realm retrieval of device attestation evidence

I_{KNHFM} A Realm is expected to retrieve cached device attestation evidence from the Host via an RSI_HOST_CALL interface. For details, refer to *Realm Host Interface specification* [20].

I_{WBRKD} A Realm can retrieve digests of the most recent device attestation evidence received by the RMM, by executing the RSI_VDEV_GET_INFO command.

See also:

- A9.5.4 *Host caching of device attestation evidence*
- B5.4.5 *RSI_HOST_CALL command*
- B5.4.19 *RSI_VDEV_GET_INFO command*

A9.6.2 Realm validation of device memory mappings

I_{GWRDH} 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 MMIO Reporting Offset (MRO) 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.

  Note that this is reported via an Arm-specific mechanism, described in *RME system architecture spec* [15].

I_{LXTVZ} A Realm can validate by execution of RSI_VDEV_VALIDATE_MAPPING that each MMIO region with IS_NON_TEE_MEM=0 in the Realm device interface report has been correctly mapped into the Realm's Protected IPA space.

S_{JCXMF} In order to determine the IPA to pass into RSI_VDEV_VALIDATE_MAPPING, the Realm uses the contents of the device interface report to calculate an offset which it adds to the virtual BAR address.

The ranges described in the device interface report may not cover the entire BAR. This can be due to the presence of region within the BAR which is TEE=0, and is therefore omitted from the device interface report.

Consequently, in order to calculate the virtual BAR offset, the Realm applies the following algorithm:

- Let `(bar_base, bar_size)` be the IPA range spanned by the virtual BAR. `bar_size` is guaranteed by PCIe to be a power of two.
- Let `(region_base, region_size)` be an entry read from the device interface report. The device generated these pseudo-PA values by adding the MRO to the corresponding system physical address.
- Use the BAR size as an input to a modulus operation to calculate the offset of the region within the virtual BAR: `region_offset = region_base % bar_size`

This algorithm assumes that the MRO value is a multiple of the BAR size.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

U_{PFCTN} Arm recommends that the RMM should compute the MMIO Reporting Offset for a given VDEV using the following formula: `MRO = x * 2^n`

Where:

- `x` is a pseudo-random number.
- `2^n` is guaranteed to be greater than or equal to the maximum BAR size for this VDEV.

Possible strategies for choosing `2^n` include the following:

- Query PCIe configuration space to determine the maximum BAR size for this VDEV.
- Compute the maximum size among the device non-coherent address ranges (ncoh_addr_range) and device coherent address ranges (coh_addr_range) of the parent PDEV.
- Use the size of the platform's MMIO region.

I_{CLXRC} The input values of RSI_VDEV_VALIDATE_MAPPING include the lock nonce, measurement sequence and report sequence values retrieved from the earlier call to RSI_VDEV_GET_INFO. These are used to ensure that the configuration of the device has not changed between verification of device attestation evidence, and validation of device memory mappings.

I_{NHQTB} 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.

I_{XSKML} Arm recommends that RSI_VDEV_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* [16].

I_{MGCSD} Execution of RSI_VDEV_VALIDATE_MAPPING initiates a VDEV mapping validation request. On execution of RSI_VDEV_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_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 RIPAS_DEV.

R_{QNNVB} The RTTE attributes checked by RMI_RTT_DEV_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 RIPAS of the target region of IPA space is RIPAS_EMPTY.
- The PA is within a PDEV address range which is consistent with the "coh" flag provided by the Realm.
  - If coh == RSI_DEV_MEM_COHERENT then the PA is expected to be within pdev.coh_addr_range.
  - If coh == RSI_DEV_MEM_NON_COHERENT then the PA is expected to be within pdev.ncoh_addr_range.
- The limited ordering properties of the PA (LOR or non-LOR) are consistent with the "order" flag provided by the Realm.

If the above checks pass then the following RTTE attributes are set:

- Memory attributes are set according the value of the "coh" flag provided by the Realm.
  - If coh == RSI_DEV_MEM_COHERENT then:
    - On platforms where it is safe, memory attributes are MEMATTR_PASSTHROUGH.
    - Otherwise, memory attributes are MEMATTR_CACHEABLE.
  - If coh == RSI_DEV_MEM_NON_COHERENT then memory attributes are MEMATTR_NON_CACHEABLE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_YVRRV Creation and validation of device memory mappings is illustrated in the following sequence diagram.

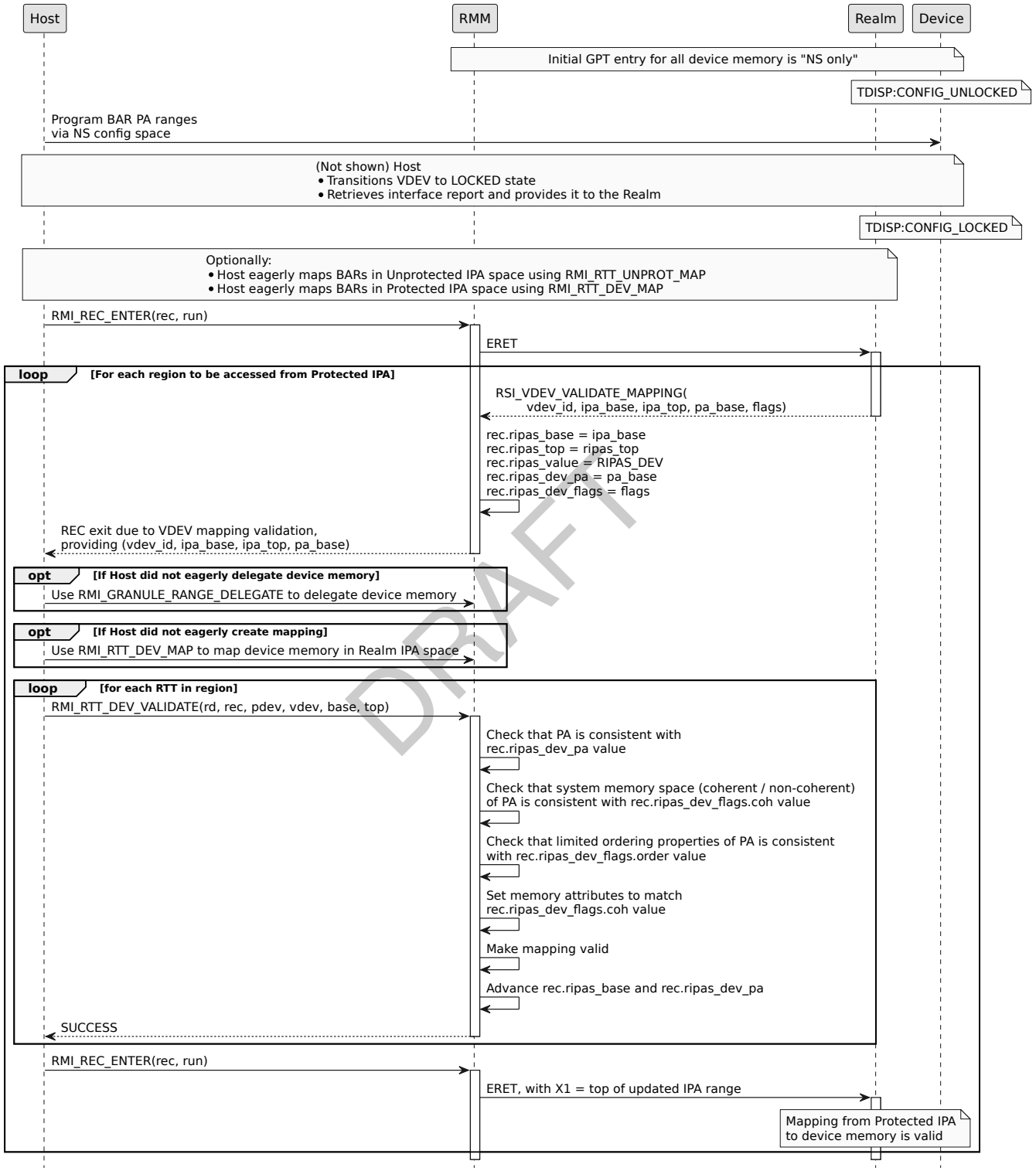


Figure A9.16: Creation and validation of device memory mappings

See also:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- *PCI Express 6.0 specification* [16]
- A5.5 *VDEV mapping validation*
- A5.6.12 *Memory attributes*
- B4.5.63 *RMI_RTT_DEV_VALIDATE command*
- B5.4.20 *RSI_VDEV_VALIDATE_MAPPING command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.6.3 Realm enablement of device DMA

I_{CWTNT} A Realm can enable device DMA by calling RSI_VDEV_DMA_ENABLE.

I_{CDBSK} The input values of RSI_VDEV_DMA_ENABLE include the lock nonce, measurement sequence and report sequence values retrieved from the earlier call to RSI_VDEV_GET_INFO. These are used to ensure that the configuration of the device has not changed between verification of device attestation evidence, and enablement of device DMA.

I_{CPMFZ} RSI_VDEV_DMA_ENABLE fails unless the state of the VDEV is VDEV_STARTED.

See also:

- B5.4.18 *RSI_VDEV_DMA_ENABLE command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.7 Physical SMMU

I_{DSBWQ} A physical device which contains functions that can be assigned to Realms must access memory via a physical SMMU (PSMMU).

I_{HMYPL} A PSMMU is identified by the base physical address of SMMUv3_PAGE_0 for the Non-secure SMMU instance.

See also:

- *Arm System Memory Management Unit Architecture Specification* [22]
- A9.8 *Virtual SMMU*

A9.7.1 PSMMU object

A9.7.1.1 PSMMU attributes

D_{HVHDP} The attributes of a PSMMU are summarized in the following table.

Name	Type	Description
state	RmmPsmmuState	State of the PSMMU
sid_size	UInt64	StreamID size in bits. This is read from the SMMU_IDR1.SIDSIZE register field.
irq_cfg	RmmIrqCfg	Interrupt configuration for Gerror, EventQ and PRIQ interrupts
cmdq_sync_irq_wired	RmmFeature	Whether the cmdq_sync interrupt is wired
gerror_intr_num	UInt32	Gerror wired interrupt number
eventq_intr_num	UInt32	EventQ wired interrupt number
priq_intr_num	UInt32	PRIQ wired interrupt number
cmdq_sync_intr_num	UInt32	CMD sync wired interrupt number
feat_ats	RmmFeature	Whether the PSMMU supports ATS
feat_pri	RmmFeature	Whether the PSMMU supports PRI
msi_config	RmmSmmuMsiConfig	MSI configuration
ppr_pending	RmmBoolean	Whether a PRI Page Request is pending on this PSMMU

DRAFT

I_{PHQHP} At platform boot, the state of a PSMMU is INACTIVE.

I_{ZHMCN} The state of a PSMMU is transitioned from INACTIVE to ACTIVE by execution of RMI_PSMMU_ACTIVATE.

I_{CXXXS} The state of a PSMMU is transitioned from ACTIVE to INACTIVE by execution of RMI_PSMMU_DEACTIVATE.

See also:

- B4.5.29 *RMI_PSMMU_ACTIVATE command*
- B4.5.30 *RMI_PSMMU_DEACTIVATE command*

A9.7.1.2 PSMMU lifecycle

A9.7.1.2.1 States

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

D_{HLXZG} The states of a PSMMU are listed below.

State	Description
PSMMU_INACTIVE	PSMMU has not been activated.
PSMMU_ACTIVE	PSMMU has been activated.

A9.7.2 PSMMU activation

R_{GYZJR} On execution of RMI_PSMMU_ACTIVATE, if the implementation requires memory to be donated then the "donating memory to a Stateful RMI Operation (SRO)" flow is followed.

U_{RKXRV} The memory required by the implementation for management of PSMMU includes the Level 1 Stream Table and queues.

I_{PHRKP} If ATS is enabled for the PSMMU, RMI_PSMMU_ACTIVATE fails unless the Level 0 DPT has been fully created before the RMI_PSMMU_ACTIVATE is called.

I_{XMWTB} For a PSMMU which supports MSI, execution of RMI_PSMMU_ACTIVATE initializes the values of the MSI address and data attributes.

See also:

- A9.7.5 *Device Permission Table*
- B4.3.2.2 *Donating memory to an SRO*
- B4.5.29 *RMI_PSMMU_ACTIVATE command*
- B4.5.32 *RMI_PSMMU_INFO command*

A9.7.3 PSMMU deactivation

R_{CPYLN} On execution of RMI_PSMMU_DEACTIVATE, if memory was donated when the PSMMU was activated then it is reclaimed via the "reclaiming memory from a Stateful RMI Operation (SRO)" flow.

See also:

- A9.7.5 *Device Permission Table*
- B4.3.2.3 *Reclaiming memory from an SRO*
- B4.5.30 *RMI_PSMMU_DEACTIVATE command*

A9.7.4 PSMMU Stream Tables

I_{PTQKF} A PSMMU Stream Table is associated with a given PSMMU instance.

I_{PGRCV} A PSMMU Stream Table has up to two levels. Level 2 is required if the PSMMU implements more than 64 Stream IDs.

I_{LJHVM} Both levels of a PSMMU Stream Table are allocated at runtime from memory which has been delegated to the RMM from DRAM.

I_{JBPSY} The size of a PSMMU Level 1 Stream Table is determined by its sid_size and split attributes and may span multiple Granules.

I_{JNXJL} The size of a PSMMU Level 2 Stream Table is one Granule.

I_{DNLDK} A PSMMU Level 2 Stream Table is created by execution of RMI_PSMMU_ST_L2_CREATE.

R_{KQMCS} On execution of RMI_PSMMU_ST_L2_CREATE, the "donating memory to a Stateful RMI Operation (SRO)" flow is followed.

I_{NNQRD} A PSMMU Level 2 Stream Table is destroyed by execution of RMI_PSMMU_ST_L2_DESTROY.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{PBDGB} On execution of RMI_PSMMU_ST_L2_DESTROY, the “reclaiming memory from a Stateful RMI Operation (SRO)” flow is followed.

D_{RMSLJ} A Level 1 Stream Table is *live* if any of its entries points to a Level 2 Stream Table.

I_{XLSCJ} The function `PsmmuL1StIsLive()` is used to evaluate whether a Level 1 Stream Table is live.

I_{RJQFD} Execution of RMI_PSMMU_DEACTIVATE fails if the Level 1 Stream Table is live.

D_{NMXVY} A Level 2 Stream Table is *live* if any of its entries corresponds to a VDEV.

I_{MZFYF} The function `PsmmuL2StIsLive()` is used to evaluate whether a Level 2 Stream Table is live.

I_{YBBPX} Execution of RMI_PSMMU_ST_L2_DESTROY fails if the Level 2 Stream Table is live.

See also:

- A9.7.2 *PSMMU activation*
- A9.7.3 *PSMMU deactivation*
- B3.88 *PsmmuL1StIsLive function*
- B3.89 *PsmmuL2StIsLive function*
- B4.3.2.2 *Donating memory to an SRO*
- B4.3.2.3 *Reclaiming memory from an SRO*
- B4.5.34 *RMI_PSMMU_ST_L2_CREATE command*
- B4.5.35 *RMI_PSMMU_ST_L2_DESTROY command*
- B4.5.74 *RMI_VDEV_CREATE command*

A9.7.5 Device Permission Table

I_{HCKWZ} A DPT is a table, indexed by physical address, which indicates the VMID(s) which are permitted to access the corresponding physical location. Its purpose is to check that accesses to Realm PAS made by a device which stores translated addresses are permitted.

I_{GGRKR} A DPT is a system-wide singleton.

I_{NKZXJ} A DPT is a two-level data structure.

I_{WFQTC} Both levels of a DPT are allocated at runtime from memory which has been delegated to the RMM.

I_{BVCCQ} The DPT Granule size is equal to the RMM Granule size.

I_{BKYKN} A Level 0 DPT is created by execution of RMI_DPT_L0_CREATE.

R_{HLHSG} On execution of RMI_DPT_L0_CREATE, the “donating memory to a Stateful RMI Operation (SRO)” flow is followed.

R_{KKKPP} Memory which is donated for the Level 0 DPT must satisfy alignment requirements specified in the SMMU architecture.

I_{DRYGV} A Level 0 DPT is destroyed by execution of RMI_DPT_L0_DESTROY.

R_{WFDBB} On execution of RMI_DPT_L0_DESTROY, the “reclaiming memory from a Stateful RMI Operation (SRO)” flow is followed.

I_{TDNQM} A Level 1 DPT is created by execution of RMI_DPT_L1_CREATE.

R_{FWYHN} On execution of RMI_DPT_L1_CREATE, the “donating memory to a Stateful RMI Operation (SRO)” flow is followed.

I_{FVQLG} A Level 1 DPT is destroyed by execution of RMI_DPT_L1_DESTROY.

R_{JWZSW} On execution of RMI_DPT_L1_DESTROY, the “reclaiming memory from a Stateful RMI Operation (SRO)” flow is followed.

S_{GVSDH} Creation or destruction of Level 1 DPTs which describes different physical address regions may be performed concurrently.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{RKQJQ} If ATS is enabled for a Realm then allocation of memory to the Realm by execution of RMI_RTT_DATA_MAP_INIT or RMI_RTT_DATA_MAP fails if the corresponding Level 1 DPT has not been created .

S_{DGJKV} RMI_DPT_L1_CREATE allows the Host to utilize NUMA awareness when allocating an L1DPT.

See also:

- *Arm System Memory Management Unit Architecture Specification* [22]
- B4.3.2.2 *Donating memory to an SRO*
- B4.3.2.3 *Reclaiming memory from an SRO*
- B4.5.1 *RMI_DPT_L0_CREATE command*
- B4.5.2 *RMI_DPT_L0_DESTROY command*
- B4.5.3 *RMI_DPT_L1_CREATE command*
- B4.5.4 *RMI_DPT_L1_DESTROY command*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*

A9.7.6 PSMMU interrupts

I_{0004} For a PSMMU the interrupt configuration is returned through RMI_PSMMU_INFO.

arm-impl-note: RMM implementation through RMI_PSMMU_INFO indicates if the realm-side SMMU needs wired, MSI or no-interrupts for its functioning.

I_{0005} For a PSMMU which supports wired interrupts, the Gerror, EventQ, PRIQ, and CMDQ-sync interrupts numbers are returned in RMI_PSMMU_INFO.

I_{0006} If the RMI_PSMMU_INFO returns Wired IRQ configuration, then Host performs wired interrupt configuration and calls RMI_PSMMU_ACTIVATE with IRQ configuration set to IRQ_WIRED.

I_{TFFVJ} For a PSMMU activated with IRQ_WIRED configuration, it is assumed that the RMM implementation will initialise SMMU_*IRQ_CFG0 registers to 0 to allow wired interrupts.

I_{YLCKP} If the RMI_PSMMU_INFO returns MSI IRQ configuration, then execution of RMI_PSMMU_ACTIVATE indicates the IRQ configuration to be IRQ_MSI and provides the MSI configuration.

I_{JTHTT} On taking a physical SMMU interrupt, the Host notifies the RMM by execution of RMI_PSMMU_IRQ_NOTIFY.

S_{PDZVW} The Host is expected to call RMI_PSMMU_IRQ_NOTIFY in a loop until RmiPsmmuIrqOutputFlags::pending is RMI_FALSE.

See also:

- A9.8.2 *VSMMU lifecycle*
- B4.5.29 *RMI_PSMMU_ACTIVATE command*
- B4.5.33 *RMI_PSMMU_IRQ_NOTIFY command*

A9.7.7 PSMMU interrupt handling

This section describes how PSMMU interrupts are handled.

I_{0007} When a PSMMU interrupt is triggered, the Host executes RMI_PSMMU_IRQ_NOTIFY command with the set of IRQs that are pending.

I_{0008} During execution of RMI_PSMMU_IRQ_NOTIFY, an RMM implementation is required to check only the interrupts that are specified in the call.

Handling of a PSMMU IRQ is described below.

1. Physical SMMU triggers an IRQ

2. Host informs RMM about the interrupts that are triggered

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Host invokes RMI_PSMMU_IRQ_NOTIFY to infom RMM of the interrupts that are triggered. The return values from this command include flags which inform the Host of the action which needs to be taken next.

The following sections describe each flag.

2a) RMM handling is pending

If flags.pending is set, the Host calls RMI_PSMMU_IRQ_NOTIFY again until flags.pending is cleared.

2b) Reports an SMMU event

To report an SMMU event to the Host RMM sets, flags.irq_event = RMI_PSMMU_IRQ_EVENT_PSMMU. This is typically used to inform SMMU events to the Host for bookkeeping or optional handling of device errors. For an SMMU event, the event number and the affected SID with syndrome information is provided in accordance with the SMMU specification.

Once the Host has finished any optional handling, Host calls RMI_PSMMU_EVENT_CONSUME command, and RMM consumes the event.

2c) Reports a virtual SMMU event

To report a virtual SMMU event to the Host RMM sets, flags.irq_event = RMI_PSMMU_IRQ_EVENT_VSMMU. If this flag is set, then the Host calls RMI_VSMMU_EVENT_NOTIFY, passing the VSMMU along with the corresponding RD, PDEV and VDEV objects.

This return value is supported only for implementations that support VSMMU feature and for VDEVs that are created with vsmmu flag.

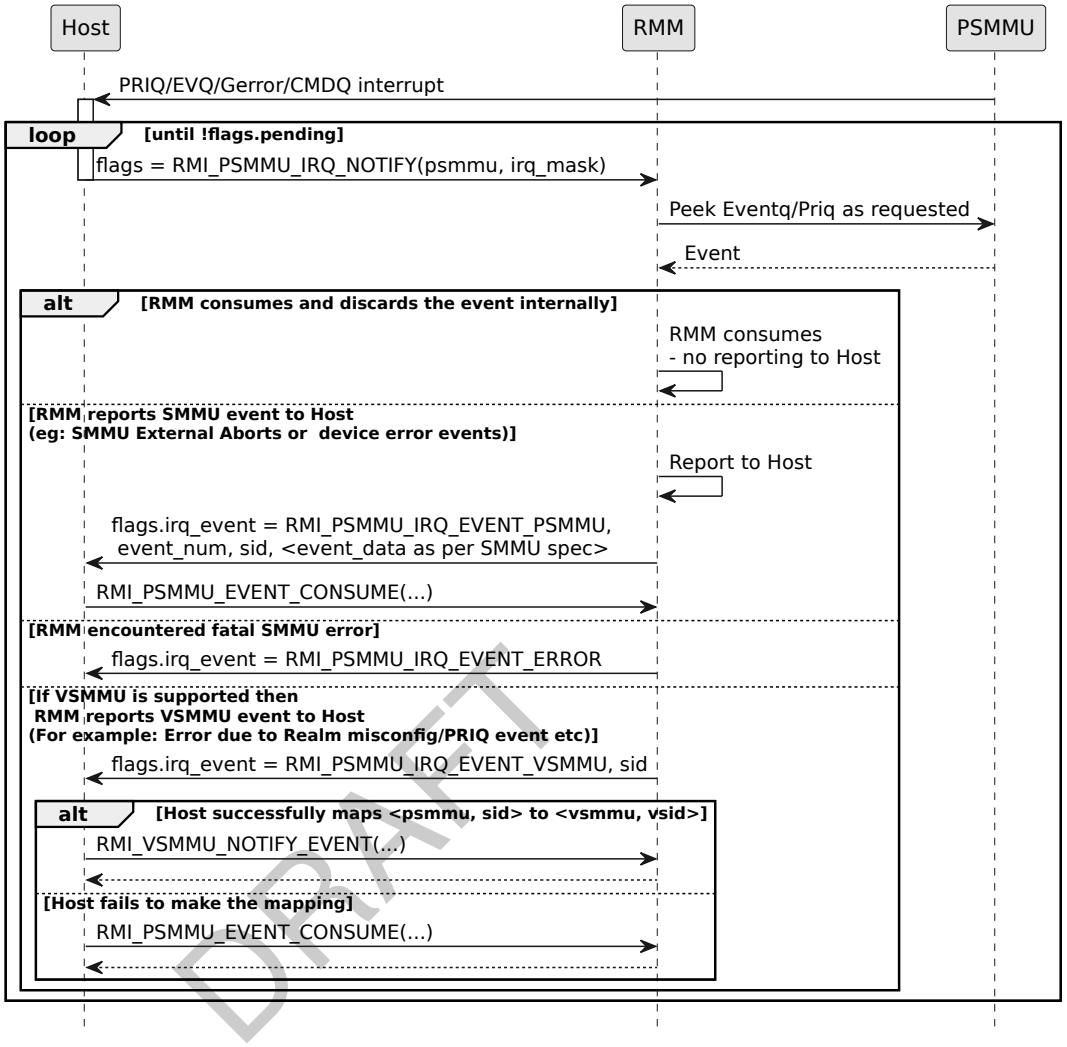
2d) Fatal SMMU error

To report a fatal SMMU error to the Host RMM sets, flags.irq_event = RMI_PSMMU_IRQ_EVENT_ERROR. This return value is used to report a fatal error in the SMMU. Following this error implementations are permitted to abort all Realm SMMU transactions.

Handling of a PSMMU IRQ is illustrated in the following sequence diagram.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



DRAFT

Figure A9.17: Handling of a PSMMU interrupt

See also:

- B4.5.29 *RMI_PSMMU_ACTIVATE command*
- B4.5.32 *RMI_PSMMU_INFO command*
- B4.5.33 *RMI_PSMMU_IRQ_NOTIFY command*
- B4.5.74 *RMI_VDEV_CREATE command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.8 Virtual SMMU

D_{LWQCC} A *Virtual SMMU* (VSMMU) object stores the state of a Arm VSMMU which is emulated by the RMM.

I_{TQGHJ} 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.

I_{KTGRM} A VSMMU can only be created while the Realm state is NEW.

I_{PSVBC} 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.

I_{GGQGB} Supported VSMMU features can be discovered by calling RMI_VSMMU_FEATURES.

I_{ZMGHD} The Host provides the VSMMU AIDR and IDR values when executing RMI_VSMMU_CREATE.

I_{GTPMC} RMI_VSMMU_CREATE fails if the provided AIDR or IDR values are not supported.

I_{TGPSW} The encoding of the AIDR and IDR values is as specified by the Non-secure world SMMU architecture, which in some cases differ from the corresponding Realm world encodings.

I_{JWYQR} A VSMMU is destroyed by execution of RMI_VSMMU_DESTROY.

I_{RQKHQ} A VSMMU is bound to a VDEV by execution of RMI_VDEV_CREATE, with RmiVdevFlags::VSMMU set.

I_{SBCYG} A VSMMU is unbound from a VDEV by execution of RMI_VDEV_DESTROY.

I_{QWBJH} Mappings from Realm Protected IPA space to a VSMMU object are created by execution of RMI_RTT_ARCH_DEV_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 HIPAS_VOID to HIPAS_ARCH_DEV.

I_{HZHZY} Mappings from Realm Protected IPA space to a VSMMU object are removed by execution of RMI_RTT_ARCH_DEV_UNMAP.

See also:

- *Arm System Memory Management Unit Architecture Specification* [22]
- A9.8.3 *VSMMU liveness*
- A9.8.4 *VSMMU validation*
- B4.5.47 *RMI_RTT_ARCH_DEV_MAP command*
- B4.5.48 *RMI_RTT_ARCH_DEV_UNMAP command*
- B4.5.74 *RMI_VDEV_CREATE command*
- B4.5.75 *RMI_VDEV_DESTROY command*
- B4.5.85 *RMI_VSMMU_CREATE command*
- B4.5.86 *RMI_VSMMU_DESTROY command*
- B4.5.88 *RMI_VSMMU_FEATURES command*

A9.8.1 VSMMU attributes

D_{PXJLR} The attributes of a VSMMU are summarized in the following table.

Name	Type	Description
state	RmmVsmmuState	State of the VSMMU
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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
aidr	Bits64	SMMU_AIDR register value
idr	Bits64[7]	SMMU_IDR register values
msi_config	RmmSmmuMsiConfig	MSI configuration

I_{LKMSG} The Host provides the VSMMU register IPA range when executing RMI_VSMMU_CREATE.

See also:

- B4.5.85 *RMI_VSMMU_CREATE command*

A9.8.2 VSMMU lifecycle

A9.8.2.1 States

D_{RLWKJ} The states of a VSMMU are listed below.

State	Description
VSMMU_INACTIVE	VSMMU has not been activated by the Realm.
VSMMU_ACTIVE	VSMMU has been activated by the Realm.

R_{LBXWV} Realm access to a VSMMU whose state is not VSMMU_ACTIVE results in an UNKNOWN exception taken to the Realm.

A9.8.2.2 State transitions

I_{YFWWL} On creation by execution of RMI_VSMMU_CREATE, the initial state of a VSMMU is VSMMU_INACTIVE.

I_{RTHHQ} Successful execution of RSI_ARCH_DEV_ACTIVATE with the base input value matching the base IPA of the VSMMU register space causes the state of the VSMMU to transition to VSMMU_ACTIVE.

R_{QQQKK} Successful execution of RSI_ARCH_DEV_ACTIVATE with the base input value matching the base IPA of the VSMMU register space causes the RIPAS of all IPAs within its MMIO interface to transition to RIPAS_DEV.

I_{WYTDS} Successful execution of RMI_RTT_ARCH_DEV_UNMAP causes the state of the VSMMU to transition to VSMMU_INACTIVE.

I_{DYYJM} Permitted VSMMU 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 VSMMU object. A transition to the pseudo-state *NULL* represents destruction of a VSMMU object.

From state	To state	Events
NULL	VSMMU_INACTIVE	RMI_VSMMU_CREATE
VSMMU_INACTIVE	VSMMU_ACTIVE	RSI_ARCH_DEV_ACTIVATE
VSMMU_ACTIVE	VSMMU_INACTIVE	RMI_RTT_ARCH_DEV_UNMAP
VSMMU_INACTIVE	*NULL*	RMI_VSMMU_DESTROY
VSMMU_ACTIVE	*NULL*	RMI_VSMMU_DESTROY

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- B4.5.48 *RMI_RTT_ARCH_DEV_UNMAP command*
- B4.5.85 *RMI_VSMMU_CREATE command*
- B4.5.86 *RMI_VSMMU_DESTROY command*
- B5.4.1 *RSI_ARCH_DEV_ACTIVATE command*

A9.8.3 VSMMU liveness

D_{HJFFC} *VSMMU liveness* is a property which means that there exists one or more mappings from Realm Protected IPA space to the VSMMU object.

I_{DTTJF} If a VSMMU is live, it cannot be destroyed.

See also:

- A9.8.2 *VSMMU lifecycle*
- B3.226 *VsmmuIsLive function*
- B4.5.86 *RMI_VSMMU_DESTROY command*

A9.8.4 VSMMU validation

I_{CWBCX} The Realm queries whether an IPA is the base address of a VSMMU by execution of RSI_VSMMU_GET_INFO.

I_{DJHHW} The Realm activates the register interface of a VSMMU by execution of RSI_ARCH_DEV_ACTIVATE. This causes the RIPAS of the IPA range to change from RIPAS_EMPTY to RIPAS_DEV.

R_{DVFYY} On execution of RSI_ARCH_DEV_ACTIVATE, if the RMM reaches an RTTE within the IPA range whose HIPAS is not HIPAS_ARCH_DEV then the command fails with RSI_ERROR_DEVICE.

I_{PNHPB} The attributes of a VSMMU are guaranteed not to change between execution of RSI_VSMMU_GET_INFO and RSI_ARCH_DEV_ACTIVATE.

I_{BYDXH} The programming model for validating and activating a VSMMU is shown in the following pseudocode:

```
int realm_validate_vsmmu(uint64_t base, uint64_t top)
{
    uint64_t new_base;
    RsiResponse 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_ARCH_DEV_ACTIVATE(base, top, &new_base);

        if (ret != RSI_SUCCESS) {
            return ret;
        }

        base = new_base;
    }

    return RSI_SUCCESS;
}
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{PWPXX} Setup of an VSMMU is illustrated in the following sequence diagram.

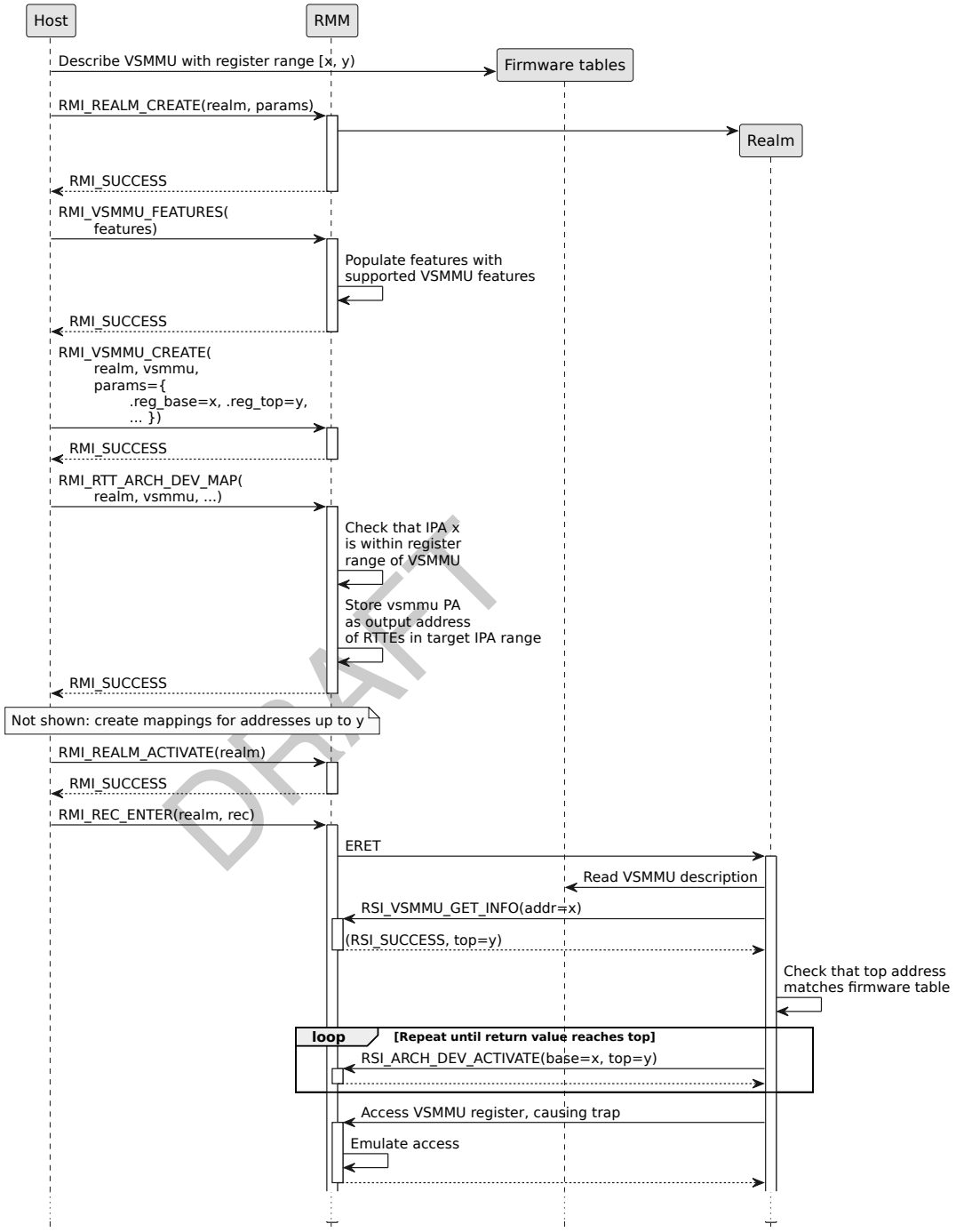


Figure A9.18: VSMMU setup

I_{JZBBJ} RSI_VDEV_GET_INFO reports to the Realm whether the VDEV is associated with a VSMMU. If so, the VSMMU base address and vSID are provided. This allows the Realm to check that the VDEV is associated with the expected VSMMU and vSID.

See also:

- A9.8.2 *VSMMU lifecycle*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B5.4.1 *RSI_ARCH_DEV_ACTIVATE command*
- B5.4.19 *RSI_VDEV_GET_INFO command*
- B5.4.22 *RSI_VSMMU_GET_INFO command*

A9.8.5 VSMMU commands

This section describes how commands provided to a VSMMU via the virtual Command Queue are handled.

I_{MNMTM} When a VSMMU Command Queue becomes non-empty due to the Realm enqueuing a command, a REC exit due to VSMMU command results.

I_{TGGMH} Handling of a VSMMU command is described below.

1. Realm enqueues command to VSMMU Command Queue

In response, the RMM performs a REC exit due to VSMMU command.

2. Host asks RMM to read VSMMU Command Queue

The Host calles RMI_VSMMU_CMD_GET, requesting the RMM to inspect the first entry in the VSMMU Command Queue.

The return values from this command include flags which inform the Host of the action which needs to be taken next. The following sections describe each flag.

2a) Complete command on PSMMU

If the “cmd_complete” flag is set then the Host calls RMI_VSMMU_CMD_COMPLETE, passing the VSMMU along with the corresponding RD, PDEV and VDEV objects.

The RMM checks the PSMMU Command Queue. If that queue has space then the RMM consumes the VSMMU Command Queue entry and enqueues a corresponding entry into the PSMMU Command Queue.

If the PSMMU Command Queue is full then a “busy” error is returned to the Host.

2b) Inject IRQ for CMDQ

If the “irq” flag is set then the Host injects a vIRQ into the Realm.

I_{HFBND} Handling of a VSMMU command is illustrated in the following sequence diagram.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

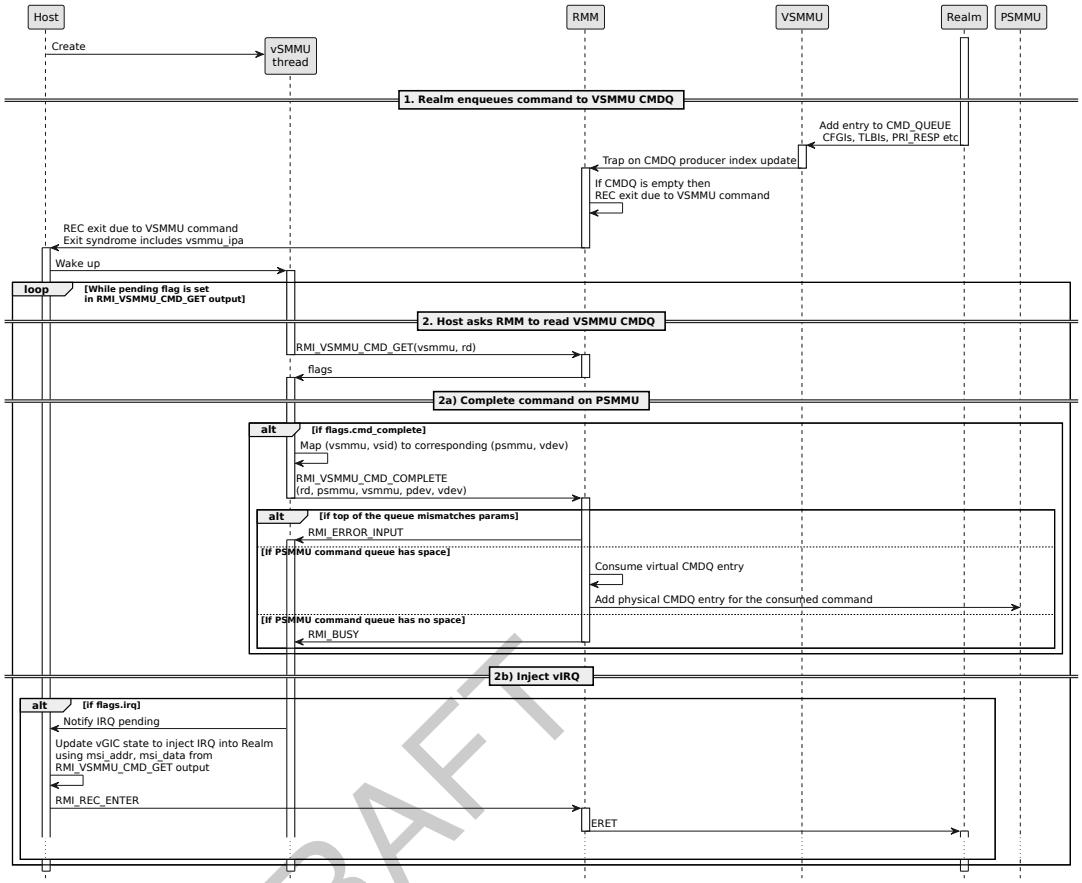


Figure A9.19: Handling of a VSMMU command

DRAFT

See also:

- A4.3.14 *REC exit due to VSMMU command*
- B4.5.83 *RMI_VSMMU_CMD_COMPLETE command*
- B4.5.84 *RMI_VSMMU_CMD_GET command*

A9.8.6 Page Request Interface events

This section describes how Page Request Interface (PRI) events are delivered to a Realm which has an assigned VSMMU.

A9.8.6.1 Page Request Interface flow

I_{RPWPB} Delivery and handling of a PRI event is described below.

1. Device sends page request to PSMMU

The device issues an ATS request, which the SMMU completes, indicating that stage 1 translation failed.

In response, the device sends a PRI Page Request (PPR) to the SMMU.

2. PSMMU recieves page request

The PSMMU enqueues the PPR into the physical SMMU PRI queue (pPRIQ).

3. If pPRIQ transitioned from empty to non-empty, PSMMU raises interrupt, which Host forwards to RMM

On taking the PRI interrupt, the Host calls the RMI_PSMMU_IRQ_NOTIFY command.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

4. RMM receives PSMMU page request

The RMM reads the PPR entry from the pPRIQ. At this point, the RMM does not advance the PSMMU consumer counter.

The RMM sets a per-PSMMU flag to indicate that a PPR is pending.

The RMM returns control to the Host, passing a flag which indicates that an event should be notified to the corresponding SMMU, and the physical Stream ID (pSID).

5. Host maps page request from PSMMU to VSMMU

If the Host fails to map the (PSMMU, pSID) tuple, it calls the RMI_PSMMU_EVENT_CONSUME command. The RMM clears the "PPR pending" flag on the PSMMU, and advances the PSMMU consumer counter. In this case, this is the end of the flow.

If the Host successfully maps the (PSMMU, pSID) tuple to the corresponding (RD, VSMMU, VDEV) values. It then calls the RMI_VSMMU_EVENT_HANDLE command, passing the latter.

6. RMM inserts page request into VSMMU

The RMM checks that the "PPR pending" flag is set on the PSMMU. It then peeks the pPRIQ to check that the top entry matches incoming parameters.

The RMM then attempts to copy the PPR into the virtual SMMU PRI queue (vPRIQ), with the SID field set to the virtual Stream ID (vSID) from the VDEV provided by the Host. This copy operation has the following possible outcomes:

(a) The vPRIQ is not accessible because HIPAS is not HIPAS_DATA

The RMM returns the faulting IPA to the Host.

(b) The vPRIQ is not accessible because RIPAS is not RIPAS_RAM, or S2AP does not permit write access

The RMM sets SMMU_R_GERROR.PRIQ_ABT_ERR in the VSMMU.

The RMM returns control to the Host, passing a flag which indicates that a virtual IRQ should be injected into the Realm, with the MSI address and data values for the GERROR interrupt which the Realm programmed into the VSMMU.

(c) The vPRIQ has not been configured

The RMM silently drops the PPR.

The RMM returns control to the Host with no flag set.

(d) The vPRIQ is full

The RMM reads STE.PPAR to determine whether it should send a PASID value to the PSMMU.

If the PPR is marked `LAST == 0`, the RMM silently drops the PPR.

If the PPR is marked `LAST == 1`, the RMM sends CMD_PRI_RESP(SUCCESS) to the PSMMU. In response, the device may issue another PPR. If the Realm has executed before the next PPR is sent, its actions may have resulted in space becoming available in the vPRIQ.

The RMM returns control to the Host with no flag set.

(e) The PPR was successfully copied into the vPRIQ

The RMM returns control to the Host, passing a flag which indicates that a virtual IRQ should be injected into the Realm, with the MSI address and data values for the PRIQ interrupt which the Realm programmed into the VSMMU.

7. Host injects virtual interrupt into Realm

The Host determines which REC should receive the PRI vIRQ, writes List Register values into the REC entry structure, and calls RMI_REC_ENTER.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

8. Realm handles VSMMU page request

On taking the PRI interrupt, the Realm reads the PPR from the vPRIQ. It then creates the required stage 1 mapping(s) and writes CMD_PRI_RESP into the virtual SMMU Command Queue (vCMDQ). This traps to the RMM.

9. RMM receives VSMMU page response

The RMM performs a REC exit due to VSMMU command, passing the VSMMU IPA.

10. Host maps page response from VSMMU to PSMMU

The Host calls RMI_VSMMU_CMD_GET, passing the VSMMU object. The return values include the vSID.

The Host maps the (RD, vSMMU, vSID) tuple to the corresponding (PSMMU, pSID) values. It then calls the RMI_VSMMU_CMD_COMPLETE command, passing the latter.

11. RMM sends page response to PSMMU

The RMM constructs a PPR using the values saved in the REC, and attempts to write it to the physical SMMU Command Queue (pCMDQ).

If the pCMDQ is full, the RMM returns RMI_BUSY to the Host.

12. PSMMU sends page response to device

If the PPR write was successful, the PSMMU consumes the command from the pCMDQ. It sends the Page Request Group (PRG) response to the device.

The device reissues the ATS request, which the SMMU completes with success, returning the translated address to the device.

I_{RPRGP} Delivery and handling of a PRI event is illustrated in the following sequence diagram.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

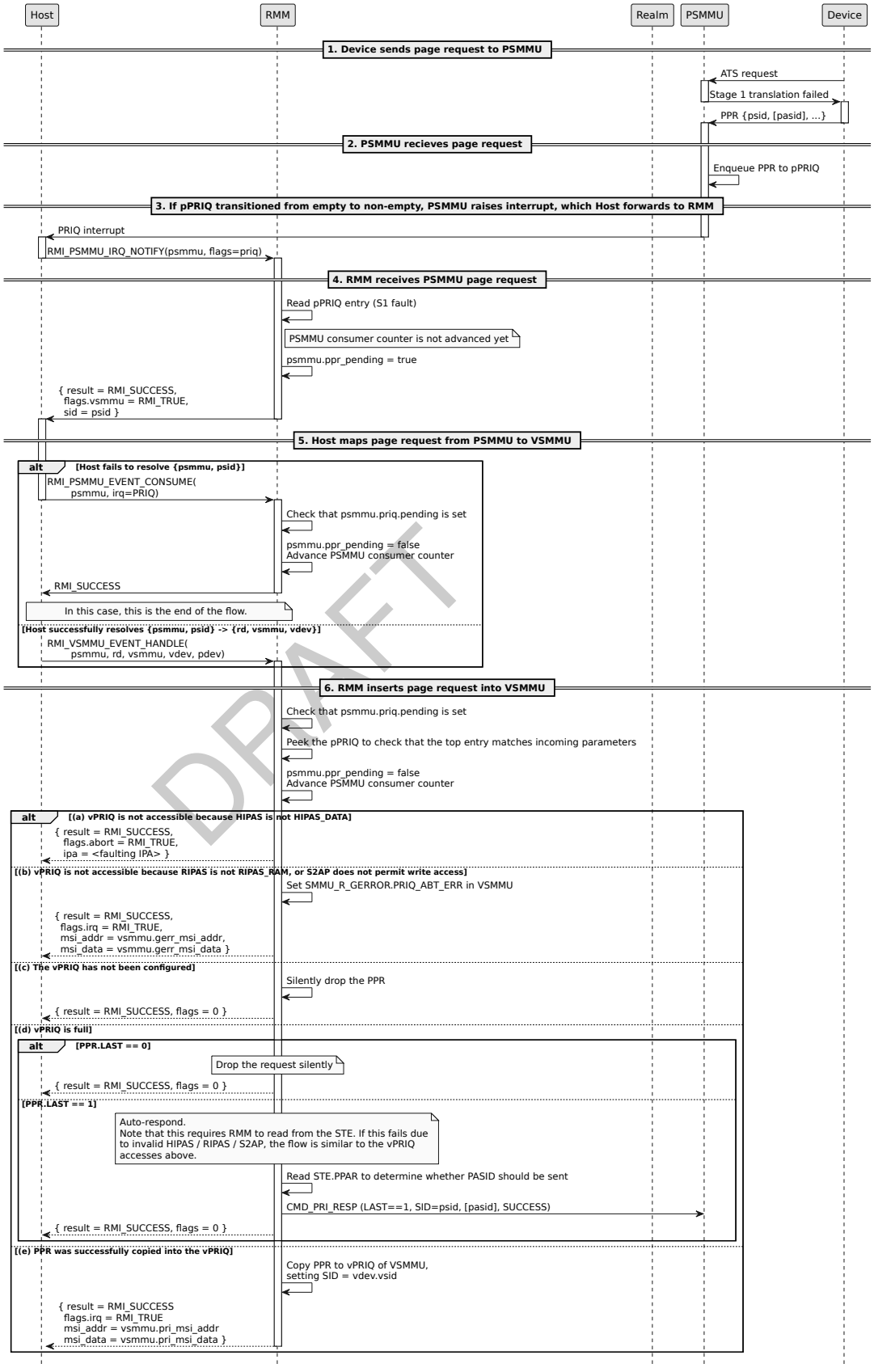


Figure A9.20: Delivery and handling of a PRI event (part 1 of 2)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

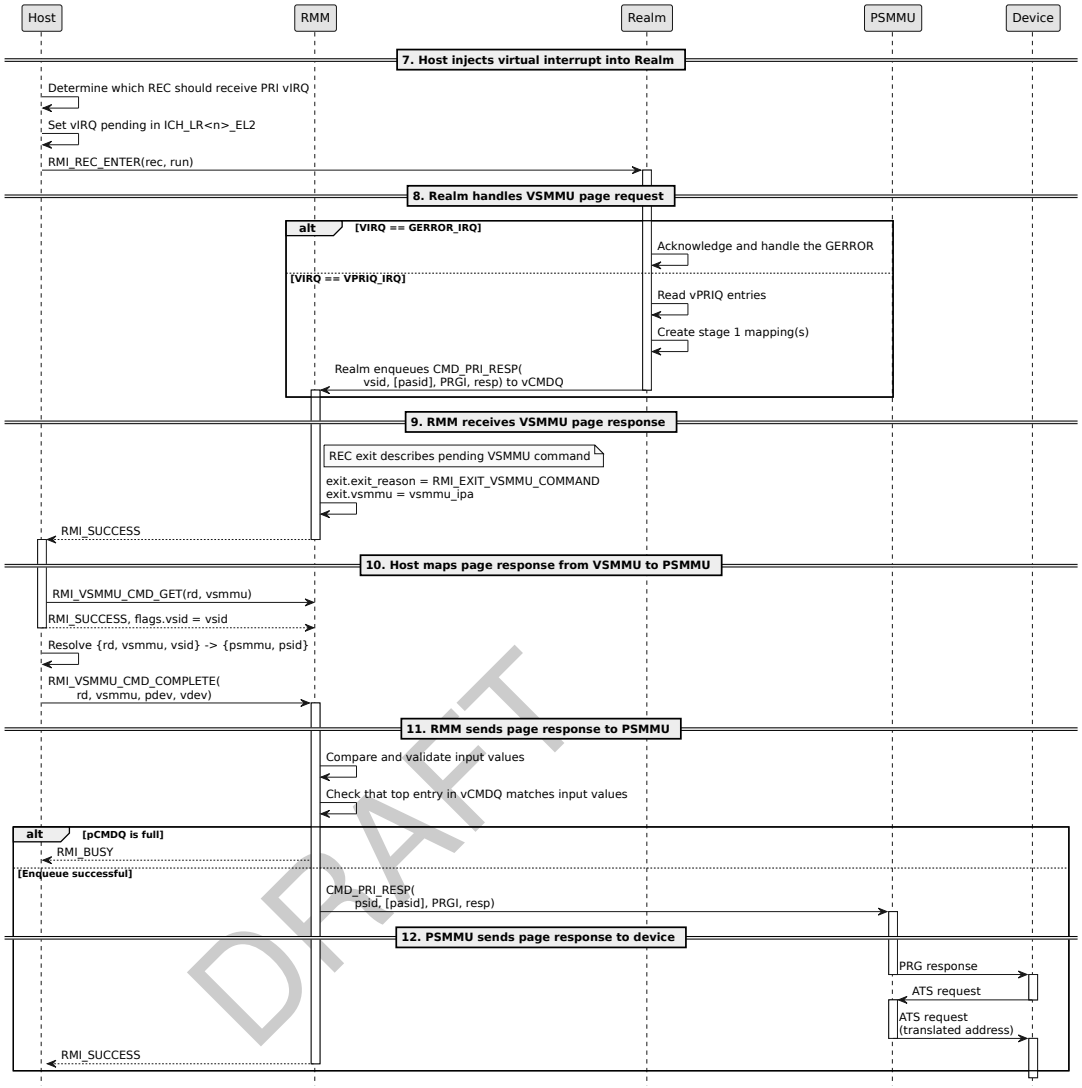


Figure A9.21: Delivery and handling of a PRI event (part 2 of 2)

See also:

- A4.3.14 *REC exit due to VSMMU command*
- B4.5.31 *RMI_PSMMU_EVENT_CONSUME command*
- B4.5.33 *RMI_PSMMU_IRQ_NOTIFY command*
- B4.5.42 *RMI_REC_ENTER command*
- B4.5.83 *RMI_VSMMU_CMD_COMPLETE command*
- B4.5.84 *RMI_VSMMU_CMD_GET command*
- B4.5.87 *RMI_VSMMU_EVENT_HANDLE command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A9.9 Device memory access

A9.9.1 Device access to a Protected IPA

R_{JMMMP} Access from a VDEV to an IPA is subject to the same stage 2 address translation used for PE accesses.

I_{JGTYM} A VDEV whose DMA state is VDEV_DMA_ENABLED is associated with a specified Plane within the Realm.

I_{SVSNG} Access from a VDEV to a Protected IPA follows the same rules, regarding stage 2 access permissions and stage 2 fault handling, as for a PE access from the associated Plane to the same IPA.

I_{PXMVS} A device may implement an Address Translation Cache (ATC), which it uses to store the result of Address Translation Service (ATS) requests issued to the SMMU.

I_{LDNFN} A device which implements an ATC may issue translated requests. In order to enforce isolation between Realms, translated requests are subject to checking by the SMMU against a Device Permission Table (DPT). The DPT is indexed by physical address, and stores the VMID which is permitted to access the corresponding physical location.

I_{HRWYF} The attributes of a Realm include the following which relate to device translated requests:

- An “ATS enable” flag which specifies whether the platform is permitted to respond to ATS requests which target the Realm’s address space.
- An “ATS Plane” value. If ATS is enabled for the Realm, this value is used by the RMM to determine which VMID to write into DPT entries corresponding to Granules which are owned by the Realm.

I_{NTMMR} The Realm can discover the values of the “ATS enable” and “ATS Plane” attributes via the RSI_REALM_CONFIG command.

I_{QRNTW} When executing the RSI_VDEV_DMA_ENABLE command, the Realm provides two parameters:

- An “ATS enable” flag.
- A “non-ATS Plane” index.

R_{QRQHB} If both the Realm “ATS enable” flag and the VDEV “ATS enable” flags are set then the device uses the stage 2 translation associated with the Realm “ATS Plane”.

Otherwise, the device uses the stage 2 translation associated with the “non-ATS Plane”.

R_{KTSKL} The STE.EATS flag is set on an SMMU stream if all of the following are true:

- The Realm “ATS enable” flag is true.
- The VDEV “ATS enable” flag is true.
- The SMMU implementation supports ATS.
- The PSMMU “ATS enable” flag is true.

See also:

- *PCI Express 6.0 specification* [16]
- *Arm System Memory Management Unit Architecture Specification* [22]
- A2.2.3 *Realm attributes*
- A5.2.3 *Realm access to a Protected IPA*
- A9.4 *Virtual device object*
- A9.7 *Physical SMMU*
- Chapter A10 *Planes*
- B5.4.16 *RSI_REALM_CONFIG command*
- B5.4.18 *RSI_VDEV_DMA_ENABLE command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A10
Planes

DRAFT

This section describes how a Realm can be divided into multiple mutually isolated execution environments, called Planes.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A10.1 Planes overview

D_{NVPCK} 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*.

I_{GRFHM} The number of auxiliary Planes is specified by the Host at Realm creation.

D_{ZDRPY} Planes within a Realm are identified using a zero-based *Plane index*.

The Plane index of the primary Plane is zero.

D_{BVDMM} 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 *Pn*.

I_{JXZMT} All Planes within a Realm share a single IPA space.

Stage 2 memory access permissions for a given IPA can differ between Planes.

I_{FMMRN} Each Plane has a VMID which is unique both within the owning Realm and among all Realms.

I_{VSJZZ} 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.

I_{BFJCV} 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.

D_{YTDGK} A given VPE executes in one Plane at a time.

This is referred to as the *active Plane* of the VPE.

I_{QXPKK} 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
- Control memory access permissions observed by other Planes in all VPEs

See also:

- A3.11 *Support for auxiliary Planes*
- A10.2 *Planes exception model*
- A10.3 *Planes memory management*
- A10.4 *Planes interrupts*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A10.2 Planes exception model

A10.2.1 Plane exception model overview

D_{GBWCV} A *Plane entry* is a transition from P0 to Pn, due to execution of RSI_PLANE_ENTER.

I_{LTWZG} P0 provides the index of the target Plane (Pn) as an input to the RSI_PLANE_ENTER command.

D_{CBVCZ} A *Plane exit* is return to P0 from an execution of RSI_PLANE_ENTER which caused a Plane entry.

D_{HBXDY} A *PlaneRun* object is a data structure used to pass values between the RMM and P0 on Plane entry and on Plane exit.

I_{CPWTD} A PlaneRun object is stored in Realm memory.

I_{ZHWPL} 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_VOID, 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 which is RIPAS_EMPTY.

   This causes a Plane exit to P0.

R_{LDDBH} Following a REC exit from P0, on the next entry to the same REC, control returns to P0.

R_{NCLJT} Following a REC exit from Pn, on the next entry to the same REC, by default control returns to Pn.

I_{TXLTM} Following a REC exit from Pn, on the next entry to the same REC, the existence of Pending virtual interrupts on the REC can cause control to return to P0, with a Plane exit due to IRQ.

R_{GPDRP} Following a REC exit from Pn, on the next entry to the same REC, if a Plane exit due to IRQ does not occur and `enter.flags.force_p0` is RMI_FORCE_P0 then control returns to P0, with a Plane exit due to Host action.

See also:

- A4.1 *Realm exception model overview*
- A10.2.3.2 *Plane exit due to IRQ*
- A10.2.3.3 *Plane exit due to Host action*
- A10.4 *Planes interrupts*
- B5.4.13 *RSI_PLANE_ENTER command*
- B5.5.17 *RsiPlaneRun type*

A10.2.2 Plane entry

D_{MDZZH} An *RsiPlaneEnter object* is a data structure used to pass values from P0 to the RMM on Plane entry.

I_{MSTNW} An RsiPlaneEnter object is stored in the RsiPlaneRun object which is passed by P0 as an input to the RSI_PLANE_ENTER command.

I_{DLWGM} In this chapter, both `plane_enter` and “the RsiPlaneEnter object” refer to the RsiPlaneEnter object which is provided to the RSI_PLANE_ENTER command.

I_{DGVWQ} On Plane entry, execution state is restored from the RsiPlaneEnter object to the PE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{KPPTJ} On Plane entry, if `plane_enter.pstate.M[3]` is set to '1' then the command fails.

I_{YNKND} On Plane entry, `SPSR_EL2` is set to the value of `plane_enter.pstate`.

I_{PQHSC} An RsiPlaneEnter object contains attributes which are used to manage Pn virtual interrupts.

D_{MFJTN} 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
pstate	`0x10`	Bits64	PSTATE
gprs[31]	`0x100`	Bits64	Registers
gicv3_hcr	`0x200`	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	`0x208`	Bits64	GICv3 List Register values
elr_el1	`0x400`	Bits64	ELR_EL1 value

A10.2.3 Plane exit

D_{BKJDY} An *RsiPlaneExit object* is a data structure used to pass values from the RMM to P0 on Plane exit.

I_{GQLBJ} An RsiPlaneExit object is stored in the RsiPlaneRun object which is passed by P0 as an input to the RSI_PLANE_ENTER command.

I_{MMZTG} In this chapter, both `plane_exit` and "the RsiPlaneExit object" refer to the RsiPlaneExit object which is provided to the RSI_PLANE_ENTER command.

I_{MPLMS} On Plane exit, execution state is saved from the PE to the RsiPlaneExit object.

D_{JYGLX} The attributes of an RsiPlaneExit object are summarized in the following table.

Name	Byte offset	Type	Description
reason	`0x0`	RsiPlaneExitReason	Exit reason
pc	`0x8`	Bits64	Program counter
pstate	`0x10`	Bits64	PSTATE
gprs[31]	`0x100`	Bits64	Registers
esr_el2	`0x200`	Bits64	Exception Syndrome Register
far_el2	`0x208`	Bits64	Fault Address Register
hpfar_el2	`0x210`	Bits64	Hypervisor IPA Fault Address register
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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
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
sctlr_el1	0x500	Bits64	SCTLR_EL1 value
vbar_el1	0x508	Bits64	VBAR_EL1 value
elr_el1	0x510	Bits64	ELR_EL1 value
pmu_ovf_status	0x600	RsiPmuOverflowStatus	PMU overflow status

I_{LPKMN} 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.

I_{MPMRX} On Plane exit, all RsiPlaneExit fields are zero unless specified otherwise.

R_{BXFJD} On Plane exit, `plane_exit.pc` contains the value of the Program Counter at the time of the Plane exit.

A10.2.3.1 Plane exit due to synchronous exception

R_{GJWRD} An exception due to any of the following in Pn causes a Plane exit due to Synchronous Exception:

- Trapped WFE instruction execution, if `plane_enter.flags.trap_wfe == RSI_TRAP`
- Trapped WFI instruction execution, if `plane_enter.flags.trap_wfi == RSI_TRAP`
- Data Abort at a Protected IPA
  - Permission fault
  - Access to an IPA which is RIPAS_EMPTY
- Instruction Abort at a Protected IPA
  - Permission fault
  - Access to an IPA which is RIPAS_EMPTY
- HVC instruction execution
- RSI_HOST_CALL execution, if `plane_enter.flags.trap_hc == RSI_TRAP`
- Access to a SIMD register or an SVE register, if `plane_enter.flags.trap_simd == RSI_TRAP`
- Any other SMC instruction execution
- A debug exception which would otherwise be taken to Pn, if `plane_enter.flags.trap_dbg == RSI_TRAP`

R_{RNVZY} Realm entry to Pn with `rec_enter.flags.inject_sea == RMI_INJECT_SEA` results in a Plane exit due to Synchronous Exception.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{LWCQY} 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 Plane 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 Plane 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 Plane exit.
- `plane_exit.pstate` contains the value of `SPSR_EL2` at the time of the Plane exit.

See also:

- A4.3.9 *REC exit due to Host call*
- A10.2.7 *Pn usage of SIMD and SVE*

A10.2.3.2 Plane exit due to IRQ

D_{DHSNV} A *Plane exit due to IRQ* is a Plane exit due to a Pending interrupt which should be handled by P0.

R_{QRMVQ} On Plane exit due to IRQ, `plane_exit.exit_reason` is RSI_EXIT_IRQ.

See also:

- A10.4 *Planes interrupts*

A10.2.3.3 Plane exit due to Host action

D_{SNFXK} A Plane exit due to Host action results from a REC entry with `enter.flags.force_p0` being set to RMI_FORCE_P0.

R_{FPRWB} On Plane exit due to Host action, all of the following are true:

- `plane_exit.exit_reason` is RSI_EXIT_HOST.
- `plane_exit.esr_el2` contains the value of `ESR_EL2` at the time of the Plane exit.
- `plane_exit.pstate` contains the value of `SPSR_EL2` at the time of the Plane exit.

See also:

- A4.2 *REC entry*

A10.2.4 REC exit from Pn

R_{HBMWH} 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 which is RIPAS_DESTROYED
  - Access to an IPA which is HIPAS_VOID and whose RIPAS is not RIPAS_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.

I_{LWSFS} Any other exception during execution of Pn causes a Plane exit to P0.

A10.2.5 Pn execution of HVC and SMC

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{PVRDD} 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.

I_{WVLCQ} 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*
- B5.4.5 *RSI_HOST_CALL command*

A10.2.6 Pn system registers

R_{LWFQV} On Realm creation, all Pn EL0 and EL1 system register values take architecturally-defined reset values.

R_{GHFLS} On Plane exit, all EL0 and EL1 system register values are saved from the PE to the REC.

R_{CLNNC} On Plane entry, all EL0 and EL1 system register values are restored from the REC to the PE. Some system register values are then overwritten with values provided by P0 to RSI_PLANE_ENTER.

U_{VFWCD} A REC must have sufficient storage for a copy of all EL0 and EL1 system register values per Plane.

I_{CSBWG} P0 can access Pn EL0 and EL1 system register values stored in the REC using the RSI_PLANE_SYSREG_READ and RSI_PLANE_SYSREG_WRITE commands.

See also:

- B5.4.14 *RSI_PLANE_SYSREG_READ command*
- B5.4.15 *RSI_PLANE_SYSREG_WRITE command*

A10.2.7 Pn usage of SIMD and SVE

R_{RTZLB} On access by Pn to a SIMD register or an SVE register, if `plane_enter.flags.trap_simd == RSI_TRAP` then a Plane exit due to Synchronous Exception occurs.

S_{HGPCK} Arm expects P0 to perform context switching of SIMD and SVE state by accessing the architectural SIMD and SVE registers.

U_{SDXSD} Arm expects the implementation to store a single copy of SIMD / SVE state in each REC, when SIMD / SVE is enabled for the parent Realm.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A10.3 Planes memory management

I_{CXSXV} All Planes within a Realm have the same IPA size.

I_{GYNKT} If a given Protected 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

I_{ZBPXQ} 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*.

D_{DLZKS} 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.

D_{VBDBB} 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

DRAFT

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).

I_{FFMLS} Within Protected IPA space, the primary RTT tree and the auxiliary RTT trees are distinct. For a given Protected IPA, the set of RTTs traversed by a walk of any auxilary RTT tree is disjoint from the set of RTTs traversed by a walk of the primary RTT tree.

I_{ZXKJN} Within Unprotected IPA space, below the RTT starting level, the primary RTT tree and the auxiliary RTT trees are shared. For a given Unprotected IPA, if a walk of any auxilary RTT tree progresses beyond the RTT starting level, then the set of RTTs traversed is identical to the set of RTTs traversed by a walk of the primary RTT tree.

I_{VDKDF} Within Protected IPA space, if a primary RTT entry is live and its state is not RTTE_TABLE, then execution of RMI_RTT_AUX_PROT_MAP creates a mapping to the same output address in an auxiliary RTT.

D_{HKQBS} A Protected IPA is *auxiliary-live* if any of the entries identified by that IPA in auxiliary RTTs are live.

I_{RHCZG} Within Unprotected IPA space, if the state of a primary RTT entry at the RTT starting level is RTTE_MAPPED_NS or RTTE_TABLE, then execution of RMI_RTT_AUX_UNPROT_MAP copies the primary RTT entry to the RTT starting level of an auxiliary RTT.

I_{LMJJG} Within Unprotected IPA space, if the state of an auxiliary RTT entry is RTTE_TABLE then its output address is an RTT in the primary RTT tree.

D_{MJTQM} Within Unprotected IPA space, a primary RTT is *auxiliary-referenced* if it is pointed to by an auxiliary RTT entry whose state is RTTE_TABLE.

I_{JWCTD} 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

U_{CPFSG} 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.

I_{BCMQH} 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_RTT_DATA_UNMAP
- RMI_RTT_DEV_UNMAP
- RMI_RTT_SET_RIPAS

I_{VWMVR} If an IPA is auxiliary-live then its RIPAS cannot be changed.

X_{VSGZB} Specifying that RIPAS is invariant while an IPA is auxiliary-live avoids an implementation of RMI_RTT_SET_RIPAS or RMI_RTT_DEV_VALIDATE having to walk multiple auxiliary RTT trees.

I_{VDDHB} Folding of primary RTTs is independent of folding of auxiliary RTTs for the same IPA range.

See also:

- A5.6.8 *RTTE liveness and RTT liveness*
- B4.5.49 *RMI_RTT_AUX_CREATE command*
- B4.5.50 *RMI_RTT_AUX_DESTROY command*
- B4.5.51 *RMI_RTT_AUX_FOLD command*
- B4.5.52 *RMI_RTT_AUX_PROT_MAP command*
- B4.5.53 *RMI_RTT_AUX_PROT_UNMAP command*
- B4.5.54 *RMI_RTT_AUX_UNPROT_MAP command*
- B4.5.55 *RMI_RTT_AUX_UNPROT_UNMAP command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.67 *RMI_RTT_SET_RIPAS command*

DRAFT

A10.3.2 Stage 2 Access Permissions within a multi-Plane Realm

This section describes how S2AP is controlled within a multi-Plane Realm.

A10.3.2.1 Stage 2 Access Permissions within a multi-Plane Realm overview

I_{FYBZM} The S2AP which applies to an access from Pn to a Protected IPA is controlled by P0.

I_{PYJGZ} The S2AP which applies to an access from Pn to a Protected IPA can be different for each Pn within the Realm.

I_{XPMCD} An access by Pn to a Protected IPA which violates S2AP causes a Plane exit taken to P0.

I_{NMMBQ} The S2AP which applies to a Realm access to an Unprotected IPA is the same for all Planes within the Realm.

I_{KYZHF} A data access by Pn to an Unprotected IPA which violates S2AP causes a REC exit taken to the Host.

I_{FWJXF} An instruction fetch by Pn to an Unprotected IPA causes a Plane exit taken to P0.

U_{LWBKP} 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*

A10.3.2.2 Stage 2 Access Permissions for a Protected IPA within a multi-Plane Realm

R_{VDNFP} The S2AP overlay index of a Protected IPA is between 0 and 14.

I_{PFKCB} S2AP overlay index 15 is RESERVED.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{PGWJC} At Realm activation, S2AP overlay indices 0 to 14 map to S2AP overlay value *RW+puX* for P0.

R_{BBYNZ} At Realm activation, S2AP overlay indices 0 to 14 map to S2AP overlay value *NoAccess* for all Planes other than P0.

D_{QKXBY} 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.

R_{ZYBWQ} At Realm activation, S2AP overlay index 0 is LOCKED.

R_{ZYKMD} At Realm activation, S2AP overlay indices 1 to 14 are UNLOCKED.

R_{BFMZG} At Realm activation, the S2AP overlay index of all Protected IPAs is 0.

I_{JFLCV} 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	*RW+puX*	*NoAccess*	LOCKED
1 to 14	*RW+puX*	*NoAccess*	UNLOCKED

I_{DFNQH} The RSI_MEM_SET_PERM_INDEX command can be used by P0 to change the S2AP overlay index for a Protected IPA.

I_{XGWMV} 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*
- B5.4.11 *RSI_MEM_SET_PERM_INDEX command*
- B5.4.12 *RSI_MEM_SET_PERM_VALUE command*

DRAFT

A10.3.2.3 Stage 2 Access Permissions change within a multi-Plane Realm

D_{PZCSW} An *S2AP change* is a process via which the S2AP of a region of Protected IPA space is changed.

I_{JPVBJ} 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 “handle”, whose initial value is zero. Across successive calls to RSI_MEM_SET_PERM_INDEX, the handle 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 RIPAS_EMPTY then at the next RMI_REC_ENTER the Host can optionally indicate that it rejects the S2AP change request.

U_{FLKBC} The purpose of the handle 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 handle value of zero.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

X_{VSQRH} 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.

I_{NMNHD} 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.

I_{RVWQP} An S2AP change is applied by one or more calls to the RMI_RTT_SET_S2AP command.

I_{FTGMY} 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.

I_{GMFPR} 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.

I_{ZHXNH} 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(uint64_t base, uint64_t top,
                   unsigned int index)
{
    uint64_t new_base;
    RsiResponse response;
    uint64_t handle = 0, new_handle;
    int ret = RSI_SUCCESS;

    while (base != top) {
        ret = rsi_mem_set_perm_index(base, top, index, handle,
                                     &new_base, &response,
                                     &new_handle);

        if (ret != RSI_SUCCESS) {
            return ret;
        }

        if (response == RSI_RESPONSE_REJECT) {
            return RSI_ERROR_INPUT;
        }

        base = new_base;
        handle = new_handle;
    }

    return RSI_SUCCESS;
}
```

I_{KVDQT} 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(uint64_t base, uint64_t top)
{
    uint64_t 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;
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            } else if (ret != RMI_SUCCESS) {
                break;
            }

            base = out_top;
        }

        return ret;
}
```

R_{RMKSF} On REC entry following a REC exit due to S2AP change, `rec.s2ap_response` is set to the value of `enter.flags.s2ap_response`.

I_{LZLPG} 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*
- B3.104 *RecS2APResponseToRsi function*
- B4.5.42 *RMI_REC_ENTER command*
- B4.5.68 *RMI_RTT_SET_S2AP command*
- B5.4.11 *RSI_MEM_SET_PERM_INDEX command*
- D1.5.4 *S2AP change flow*

A10.3.2.4 Stage 2 Access Permissions reset due to Host action

I_{TKGGN} Execution of RMI_RTT_DESTROY or RMI_RTT_AUX_DESTROY sets the S2AP overlay index of the target RTTE to 0.

X_{VKFHM} 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 RIPAS_RAM. Therefore, this specification defines the value to which the S2AP overlay index must be reset on a RIPAS transition to RIPAS_RAM.

See also:

- A5.4 *RIPAS change*

A10.3.2.5 Stage 2 Access Permissions reset due to Realm action

I_{PTYQX} For a Realm which is configured to have an RTT tree per Plane, execute permission for Pn is dropped on transition to RIPAS_DEV.

I_{QQVZH} Following a transition from RIPAS_DEV to RIPAS_EMPTY, in order to ensure that a subsequent instruction fetch from Pn does not result in a permission fault, P0 must restore execute permission by executing RSI_MEM_SET_PERM_INDEX.

See also:

- B5.4.11 *RSI_MEM_SET_PERM_INDEX command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A10.4 Planes interrupts

I_{NWWJR} On REC creation, the GIC owner for the REC is P0.

I_{PHNVC} On Plane entry, P0 can transfer GIC ownership to the target Pn.

I_{LCZHR} On Plane entry, if P0 transfers 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.

S_{TQWYC} 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.

I_{QCHTS} P0 can read the value of ICH_VTR_EL2 using the RSI_REALM_CONFIG command.

I_{MSZYK} 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.

I_{CHMCX} On Plane exit, Pn’s GIC state is exposed to P0 via the RsiPlaneExit object.

R_{ZYLRD} On REC entry the GIC state provided by the Host is assigned to the GIC owner.

R_{YXJHR} On REC entry, if the values of ICH_LR_EL2 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 IRQ.

R_{QZFYT} On REC entry, if all of the following is true then control returns to P0, resulting in a Plane exit due to IRQ:

- 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.

I_{GJRBD} On REC exit, the Realm GIC state of the GIC owner Plane is reported to the Host.

R_{HCKNY} During execution of a Plane which is not the GIC owner, direct injection of interrupts is masked.

U_{JLJKV} Direct injection of interrupts can be masked by setting `ICH_HCR_EL2.DVIM = 1`.

R_{RDFSK} During execution of the GIC owner Plane, it is IMPLEMENTATION DEFINED whether direct injection of interrupts is masked.

See also:

- A2.4.2 *REC attributes*
- A4.1 *Realm exception model overview*
- A6.1 *Realm interrupts*
- A10.2.3.2 *Plane exit due to IRQ*
- B5.4.13 *RSI_PLANE_ENTER command*
- B5.4.16 *RSI_REALM_CONFIG command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A10.5 Planes timers

R_{WMQCZ} On REC exit from P0, the Realm EL1 timer state reported to the Host is P0's EL1 timer state.

D_{PLQSG} A Realm EL1 timer is *active* if it is enabled and unmasked.

R_{RHHQZ} 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.

I_{VGCZC} 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

DRAFT

I_{RSSWX} On Plane exit, Pn's EL1 timer state is exposed to P0 via the RsiPlaneExit object.

S_{ZZXHD} 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.

I_{GPCHS} On Plane entry, the RMM may mask the hardware timer signal, following the same logic as for REC entry.

U_{NMJPL} 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*
- B5.5.15 *RsiPlaneExit type*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A10.6 Planes debug and performance monitoring

This section describes the debug and performance monitoring features which are available to Pn.

A10.6.1 Planes PMU

R_{XLPBV} On Plane exit, `plane_exit.pmu_ovf_status` indicates the status of the PMU overflow at the time of the Plane exit.

R_{QRPZG} The number of PMU counters available to Pn is determined by the value of RmiRealmParams::pmu_num_ctrs.

See also:

- A3.6 *Realm support for Performance Monitors Extension*
- A10.2.3 *Plane exit*

A10.6.2 Planes debug

R_{MXKYR} The number of breakpoints available to Pn is determined by the value of RmiRealmParams::num_bps.

R_{SBVPW} The number of watchpoints available to Pn is determined by the value of RmiRealmParams::num_wps.

R_{KSJNV} On a debug exception which is taken from Pn, if `plane_enter.flags.trap_dbg == RSI_TRAP` then a Plane exit due to Synchronous Exception occurs.

U_{SKSMJ} Setting `plane_enter.flags.trap_dbg == RSI_TRAP` causes the RMM to set `MDCR_EL2.TDE = '1'` during Plane entry.

See also:

- A3.5 *Realm support for self-hosted debug*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter A11
Realm memory encryption

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

A11.1 Realm memory encryption overview

D_{GGPSY} 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 conventional memory which is mapped in the Realm's Protected IPA space, and the RTTs which are owned by the Realm.

U_{XCZYB} Other Granules owned by the Realm, such as REC and RD, are protected with the RMM's MEC.

I_{XFMFV} The number of MECs supported by an implementation is reported by the RMI_FEATURES command in RmiFeatureRegister1::MEC_COUNT.

R_{PGNWN} If FEAT_MEC is either not implemented or not enabled, MEC_COUNT is zero.

U_{VGNBP} On a platform which implements FEAT_MEC, MEC_COUNT 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 `MEC_COUNT = (2 ^ MECID_WIDTH)- (NUM_RESERVED_MECIDS + 1)`

See also:

- *Arm Architecture Reference Manual Supplement, The Realm Management Extension (RME), for Armv9-A* [2]
- A3.14 *Support for Realm memory encryption*
- B4.6.24 *RmiFeatureRegister1 type*

A11.1.1 MEC and Realms

I_{HDHLC} RMI_REALM_CREATE fails if the system does not have an available MECID which satisfies the requested MEC policy.

I_{NPVPN} On a platform which reports MEC_COUNT to be zero, all Realms use a Shared MEC.

I_{NSRHY} On a platform which reports MEC_COUNT 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 MEC_COUNT.
- 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 MEC_COUNT.

I_{MNGFR} The Realm attestation token includes a claim which describes the MEC policy of the Realm.

See also:

- A7.2.3.1.8 *Realm MEC policy claim*
- B4.5.37 *RMI_REALM_CREATE command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

DRAFT

Part B
Interface

Chapter B1
Commands

DRAFT

This chapter describes how RMM commands are defined in this specification.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B1.1 Overview

R_{VZRKZ} The RMM exposes the following interfaces to the Host:

- The *Realm Management Interface* (RMI)

R_{NPLKX} The RMM exposes the following interfaces to a Realm:

- The *Realm Services Interface* (RSI)
- The *Power State Coordination Interface* (PSCI)

Any other SMC executed by a Realm returns SMCCC_NOT_SUPPORTED.

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} To determine whether an RMM interface is implemented, software should use the following flow:

1. Determine whether the SMCCC_VERSION command is implemented, following the procedure described in *Arm SMC Calling Convention* [23].
2. Check that the SMCCC version is >= 1.1.
3. Execute the <Interface>.Version command, which returns:
   - SMCCC_NOT_SUPPORTED (-1) if <Interface> is not implemented.
   - A version number (>0) if <Interface> is implemented.

R_{YBXKR} All data types defined in this specification are little-endian.

See also:

- Chapter B4 *Realm Management Interface*
- Chapter B5 *Realm Services Interface*
- Chapter B6 *Power State Control Interface*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

S_{STJHR} On calling an RMI or RSI command, any of X1 - X16 which are not specified as input values in the command definition SBZ.

See also:

- SMCCC *Arm SMC Calling Convention* [23]

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
`fid`	`X0`	[63:0]	*UInt64*	Command FID
`params_ptr`	`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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Output values

Name	Register	Field	Type	Description
result	X0	[15:0]	CommandReturnCode	Command return value
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: !AddrIsRmiGranuleAligned(params_ptr) post: ResultEqual(result, ERROR_INPUT)

Success conditions

ID	Post-condition
sum	sum == params.x + params.y
zero	zero == (params.x == 0) || (params.y == 0)

DRAFT

B1.3 Command registers

D_{ZDGNM} An *FID* is a value which identifies a particular RMM command.

I_{MJQGK} 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_{XLSFS} An *input value* is a value read by an RMM command from general-purpose registers.

D_{VCDCW} An *output value* is a value written by an RMM command to general-purpose registers.

D_{CZLVJ} A *command return code* 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 *condition expression* is an expression which evaluates to a boolean value.

I_{BNPKQ} Following expansion of macros, a *condition expression* is a valid expression in Arm Specification Language (ASL).

See also:

- *Arm Specification Language Reference Manual* [24]
- Chapter B3 *Command condition functions*

B1.5 Command context values

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

```
!AddrIsRmiGranuleAligned(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:

```
!AddrIsRmiGranuleAligned(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 GRAN_RD to GRAN_DELEGATED
- The state of the RTT base Granule, whose address was previously held in the RD, changes from GRAN_RTT to GRAN_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
`rtt_base`	Address	`Realm(rd).rtt_base`	true	RTT base address

The state change of the RTT Granule can then be expressed as:

```
Granule(rtt_base).state == GRAN_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:

- B4.5.37 *RMI_REALM_CREATE command*

B1.6 Command failure conditions

D_{DNQQC} 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

DRAFT

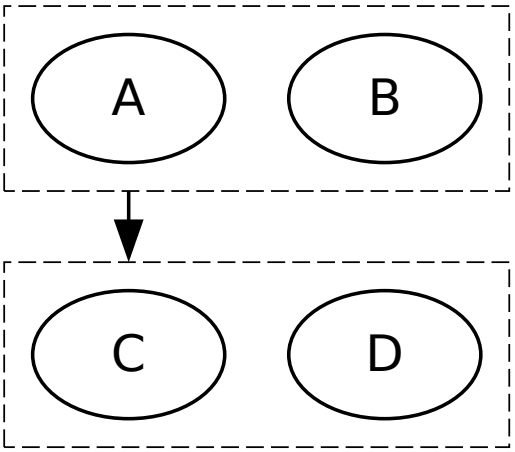


Figure B1.1

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{VHFHD} If no failure pre-condition evaluates to true then the command succeeds.

B1.7 Command success conditions

D_{SZGNZ} An RMM command *success condition* defines an observable effect of a successful execution of the command.

I_{FWCRJ} A success condition consists of a *post-condition* and an optional *pre-condition*. If the pre-condition is omitted then it is interpreted as being true.

I_{KDFKT} A success pre-condition is a condition expression whose terms can include input values and context values.

I_{LZXHB} A success post-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_{MKRVV} If an RMM command succeeds then for every success conditions whose pre-condition evaluates to true, the corresponding post-condition evaluates to true.

B1.8 Concrete and abstract types

D_{NXQWV} A *concrete type* is a type which has a defined encoding.

Examples of concrete types include:

- 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} An *abstract type* is a type which does not have a defined encoding.

Examples of abstract types include:

- 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. The type of each member of an abstract struct is an abstract type.

I_{QZRGY} Abstract types are used to model the internal state of the RMM.

I_{LMKGP} 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 `Equal()` function for each pair of corresponding concrete and abstract types.

See also:

- B3.28 *Equal function*

B1.9 Command footprint

D_{ZDJDB} The *footprint* 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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} Execution of an RMM command does not have any observable effects except for the following:

- Items in the footprint of an RMM command
- Registers in the output values of the RMM command
- Contents of memory which is writeable by the caller

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 = !AddrIsRmiGranuleAligned(params_ptr);

    // 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter B2
Interface versioning

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 != 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 $min_P < min_Q$

I_{ZCPBC} For each interface, an RMM implementation supports a set of revisions. The size of this set is at least one.

I_{RMSLZ} If an RMM implementation supports a given interface revision *(x, y)* then Arm expects that it will also supports all earlier revisons with the same major version number. That is:

(x, 0), *(x, 1)* … *(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)* … *(x, bad-1)*, *(x, bad+1)* … *(x, y-1)*, *(x, y)*.

I_{GLDQG} The set of interface revisions supported by an RMM implementation may include revisons with different major version numbers, for example:

(1, 0), *(1, 1)* … *(1, m)*

(2, 0), *(2, 1)* … *(2, n)*

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”.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 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)*

DRAFT

See also:

- B4.1 *RMI version*
- B4.5.82 *RMI_VERSION command*
- B5.1 *RSI version*
- B5.4.21 *RSI_VERSION command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter B3
Command condition functions

DRAFT

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]`.

```
pure 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.

```
pure func AddrIsAligned(
    addr : Address,
    n : integer) => boolean
begin
    var x : integer = UInt(addr);
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    return Align(x, n) == x;
end;
```

B3.3 AddrIsAuxLive function

Returns TRUE if IPA `addr` is *auxiliary-live*, that is live in any auxiliary RTT.

```
readonly func AddrIsAuxLive(
    addr : Address,
    realm : RmmRealm) => boolean
```

B3.4 AddrIsAuxRef function

Returns TRUE if the primary RTT entry at IPA `addr` is *auxiliary-referenced*, that is pointed to by an auxiliary RTT entry.

```
readonly func AddrIsAuxRef(
    addr : Address,
    realm : RmmRealm) => boolean
```

B3.5 AddrIsProtected function

Returns TRUE if address `addr` is a Protected IPA for `realm`.

```
pure func AddrIsProtected(
    addr : Address,
    realm : RmmRealm) => boolean
begin
    return UInt(addr) < 2^(realm.ipa_width - 1);
end;
```

B3.6 AddrIsRmiGranuleAligned function

Returns TRUE if address `addr` is aligned to the size of an RMI Granule.

```
readonly func AddrIsRmiGranuleAligned(
    addr : Address) => boolean
begin
    var rmm : RmmGlobal = Rmm();

    return AddrIsAligned(addr, rmm.dynamic.rmi_granule_size);
end;
```

See also:

- A2.3.1 *Granule size*

B3.7 AddrIsRsiGranuleAligned function

Returns TRUE if address `addr` is aligned to the size of an RSI Granule.

```
readonly func AddrIsRsiGranuleAligned(
    addr : Address) => boolean
begin
    return AddrIsAligned(addr, RSI_GRANULE_SIZE);
end;
```

See also:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A2.3.1 *Granule size*

B3.8 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.

```
readonly func AddrIsRttLevelAligned(
    addr : Address,
    level : integer) => boolean
```

B3.9 AddrIsTrackingRegionAligned function

Returns TRUE if address `addr` is aligned to the size of a tracking region.

```
readonly func AddrIsTrackingRegionAligned(
    addr : Address) => boolean
begin
    var rmm : RmmGlobal = Rmm();

    return AddrIsAligned(addr, rmm.dynamic.tracking_region_size);
end;
```

B3.10 AddrIsWithin function

Returns TRUE if address `addr` is within the outer range `[base, top)`.

```
pure 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;
```

B3.11 AddrRangeIsAuxLive function

Returns TRUE if any IPA in range `[base, top)` is *auxiliary-live*, that is live in any auxiliary RTT.

```
readonly func AddrRangeIsAuxLive(
    base : Address,
    top : Address,
    realm : RmmRealm) => boolean
```

B3.12 AddrRangeIsProtected function

Returns TRUE if all addresses in range `[base, top)` are Protected IPAs for `realm`.

```
readonly func AddrRangeIsProtected(
    base : Address,
    top : Address,
    realm : RmmRealm) => boolean
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
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;
```

B3.13 AddrRangeIsWithin function

Returns TRUE if all addresses in the inner range `[inner_base, inner_top)` are within the outer range `[outer_range, outer_top)`.

```
pure 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;
```

B3.14 AddrSetAllDelegableCohDevMem function

Returns TRUE if the first `size` bytes of the Address Set identified by `set_addr` and `set_addr_type` are delegable coherent device memory.

```
readonly func AddrSetAllDelegableCohDevMem(
    set_desc : RmiAddrSetDesc,
    set_addr_type : RmiRttAddrType,
    size : integer) => boolean
begin
    var rmm : RmmGlobal = Rmm();

    var offset : integer = 0;
    while offset < size do
        var addr : Address = AddrSetEntry(set_desc, set_addr_type, offset);
        if !PaIsDelegableCohDevMem(addr) then
            return FALSE;
        end;

        offset = offset + rmm.dynamic.rmi_granule_size;
    end;

    return TRUE;
end;
```

B3.15 AddrSetAllDelegableNonCohDevMem function

Returns TRUE if the first `size` bytes of the Address Set identified by `set_addr` and `set_addr_type` are delegable non-coherent device memory.

```
readonly func AddrSetAllDelegableNonCohDevMem(
    set_desc : RmiAddrSetDesc,
    set_addr_type : RmiRttAddrType,
    size : integer) => boolean
begin
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    var rmm : RmmGlobal = Rmm();

    var offset : integer = 0;
    while offset < size do
        var addr : Address = AddrSetEntry(set_desc, set_addr_type, offset);
        if !PaIsDelegableNonCohDevMem(addr) then
            return FALSE;
        end;

        offset = offset + rmm.dynamic.rmi_granule_size;
    end;

    return TRUE;
end;
```

B3.16 AddrSetEntry function

Returns address of the `offset` byte within the Address Set identified by `set_addr` and `set_addr_type`.

```
readonly func AddrSetEntry(
    set_desc : RmiAddrSetDesc,
    set_addr_type : RmiRttAddrType,
    offset : integer) => Address
```

B3.17 AlignDownToRttLevel function

Round down `addr` to align to the size of the address range described by an RTTE in a level `level` RTT.

```
readonly func AlignDownToRttLevel(
    addr : Address,
    level : integer) => Address
```

B3.18 AlignUpToRttLevel function

Round up `addr` to align to the size of the address range described by an RTTE in a level `level` RTT.

```
readonly func AlignUpToRttLevel(
    addr : Address,
    level : integer) => Address
```

B3.19 AnyRecRunning function

Returns TRUE if the state of any REC owned by `realm` is REC_RUNNING.

```
readonly func AnyRecRunning(
    realm : RmmRealm) => boolean
```

See also:

- A2.4 *Realm Execution Context*
- B4.5.39 *RMI_REALM_TERMINATE command*
- D1.2.5 *Realm destruction flow*

B3.20 AttestationTokenMaxSize function

Maximum size of attestation token in bytes.

```
readonly impdef func AttestationTokenMaxSize(
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    realm : RmmRealm) => integer
```

B3.21 AttestationTokenWrite function

Write fragment of attestation token.

Returns amount of data written in bytes.

```
readonly func AttestationTokenWrite(
    addr : Address,
    offset : integer,
    size : integer) => integer
```

B3.22 CurrentRealm function

Returns the current Realm.

```
readonly func CurrentRealm() => RmmRealm
```

B3.23 CurrentRec function

Returns the current REC.

```
readonly func CurrentRec() => RmmRec
```

B3.24 DeviceCommunicate function

Process device communication data and return the new state of the device transaction.

```
readonly func DeviceCommunicate(
    pdev : RmmPdev,
    data : RmiDevCommData) => RmmDevCommState
```

```
readonly func DeviceCommunicate(
    vdev : RmmVdev,
    data : RmiDevCommData) => RmmDevCommState
```

```
readonly func DeviceCommunicate(
    vdev : RmmVdev) => RmmDevCommState
```

B3.25 DptEntryCanDescribe function

Returns TRUE if all of the following are true:

- The L0DPT has been created
- `addr` is aligned to the size of the DPT entry for address `addr`.
- The size of the DPT entry for address `addr` is not larger than `size`.

```
readonly func DptEntryCanDescribe(
    addr : Address,
    size : integer) => boolean
begin
    var rmm : RmmGlobal = Rmm();

    // Check that L0DPT has been created
    if (DptL0().state != DPT_L0_VALID) then
        return FALSE;
    end;
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    var l0_entry : RmmDptL0Entry = DptL0Walk(addr);
    var entry_size : integer;
    case l0_entry.state of
        when DPT_L0_ENTRY_BLOCK => entry_size = rmm.static.l0dptsz;
        when DPT_L0_ENTRY_TABLE => entry_size = rmm.dynamic.rmi_granule_size;
    end;

    return (AddrIsAligned(addr, entry_size) && (entry_size <= size));
end;
```

B3.26 DptL0 function

Returns the Level 0 DPT.

This is a system-wide singleton.

```
readonly func DptL0() => RmmDptL0
```

B3.27 DptL0Walk function

Returns the Level 1 DPT which describes the physical address region starting from `addr`.

```
readonly func DptL0Walk(
    addr : Address) => RmmDptL0Entry
```

B3.28 Equal function

Check whether concrete and abstract values are equal

```
pure func Equal(
    abstract : RmmFeature,
    concrete : RmiFeature) => boolean
```

```
pure func Equal(
    concrete : RmiFeature,
    abstract : RmmFeature) => boolean
```

```
pure func Equal(
    abstract : RmmHashAlgorithm,
    concrete : RmiHashAlgorithm) => boolean
```

```
pure func Equal(
    concrete : RmiHashAlgorithm,
    abstract : RmmHashAlgorithm) => boolean
```

```
pure func Equal(
    abstract : RmmIrqCfg,
    concrete : RmiIrqCfg) => boolean
```

```
pure func Equal(
    concrete : RmiIrqCfg,
    abstract : RmmIrqCfg) => boolean
```

```
pure func Equal(
    abstract : RmmLfaPolicy,
    concrete : RmiLfaPolicy) => boolean
```

```
pure func Equal(
    concrete : RmiLfaPolicy,
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    abstract : RmmLfaPolicy) => boolean
```

```
pure func Equal(
    abstract : RmmMecPolicy,
    concrete : RmiMecPolicy) => boolean
```

```
pure func Equal(
    concrete : RmiMecPolicy,
    abstract : RmmMecPolicy) => boolean
```

```
pure func Equal(
    abstract : RmmMemCategory,
    concrete : RmiMemCategory) => boolean
```

```
pure func Equal(
    concrete : RmiMemCategory,
    abstract : RmmMemCategory) => boolean
```

```
pure func Equal(
    abstract : RmmOpCanCancel,
    concrete : RmiOpCanCancel) => boolean
```

```
pure func Equal(
    concrete : RmiOpCanCancel,
    abstract : RmmOpCanCancel) => boolean
```

```
pure func Equal(
    abstract : RmmPdevCategory,
    concrete : RmiPdevCategory) => boolean
```

```
pure func Equal(
    concrete : RmiPdevCategory,
    abstract : RmmPdevCategory) => boolean
```

```
pure func Equal(
    abstract : RmmPdevSpdm,
    concrete : RmiPdevSpdm) => boolean
```

```
pure func Equal(
    concrete : RmiPdevSpdm,
    abstract : RmmPdevSpdm) => boolean
```

```
pure func Equal(
    abstract : RmmPdevState,
    concrete : RmiPdevState) => boolean
```

```
pure func Equal(
    concrete : RmiPdevState,
    abstract : RmmPdevState) => boolean
```

```
pure func Equal(
    abstract : RmmPdevStreamType,
    concrete : RmiPdevStreamType) => boolean
```

```
pure func Equal(
    concrete : RmiPdevStreamType,
    abstract : RmmPdevStreamType) => boolean
```

```
pure func Equal(
    abstract : RmmRecRunnable,
    concrete : RmiRecRunnable) => boolean
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
pure func Equal(
    concrete : RmiRecRunnable,
    abstract : RmmRecRunnable) => boolean
```

```
pure func Equal(
    abstract : RmmRipas,
    concrete : RmiRipas) => boolean
```

```
pure func Equal(
    concrete : RmiRipas,
    abstract : RmmRipas) => boolean
```

```
pure func Equal(
    abstract : RmmState,
    concrete : RmiRmmState) => boolean
```

```
pure func Equal(
    concrete : RmiRmmState,
    abstract : RmmState) => boolean
```

```
pure func Equal(
    abstract : RmmRttPlaneFeature,
    concrete : RmiRttPlaneFeature) => boolean
```

```
pure func Equal(
    concrete : RmiRttPlaneFeature,
    abstract : RmmRttPlaneFeature) => boolean
```

```
pure func Equal(
    abstract : RmmRttS2APBase,
    concrete : RmiRttS2APBase) => boolean
```

```
pure func Equal(
    concrete : RmiRttS2APBase,
    abstract : RmmRttS2APBase) => boolean
```

```
pure func Equal(
    abstract : RmmRttS2APEncoding,
    concrete : RmiRttS2APEncoding) => boolean
```

```
pure func Equal(
    concrete : RmiRttS2APEncoding,
    abstract : RmmRttS2APEncoding) => boolean
```

```
pure func Equal(
    abstract : RmmTrackingRegionState,
    concrete : RmiTrackingRegionState) => boolean
```

```
pure func Equal(
    concrete : RmiTrackingRegionState,
    abstract : RmmTrackingRegionState) => boolean
```

```
pure func Equal(
    abstract : RmmVdevState,
    concrete : RmiVdevState) => boolean
```

```
pure func Equal(
    concrete : RmiVdevState,
    abstract : RmmVdevState) => boolean
```

```
pure func Equal(
    abstract : RmmFeature,
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    concrete : RsiFeature) => boolean
```

```
pure func Equal(
    concrete : RsiFeature,
    abstract : RmmFeature) => boolean
```

```
pure func Equal(
    abstract : RmmHashAlgorithm,
    concrete : RsiHashAlgorithm) => boolean
```

```
pure func Equal(
    concrete : RsiHashAlgorithm,
    abstract : RmmHashAlgorithm) => boolean
```

```
pure func Equal(
    abstract : RmmRipas,
    concrete : RsiRipas) => boolean
```

```
pure func Equal(
    concrete : RsiRipas,
    abstract : RmmRipas) => boolean
```

```
pure func Equal(
    abstract : RmmRipasChangeDestroyed,
    concrete : RsiRipasChangeDestroyed) => boolean
```

```
pure func Equal(
    concrete : RsiRipasChangeDestroyed,
    abstract : RmmRipasChangeDestroyed) => boolean
```

B3.29 FeatureToRmi function

Convert feature bit to RMI type.

```
readonly 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;
```

B3.30 FeatureToRsi function

Convert feature bit to RSI type.

```
readonly 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;
```

B3.31 Gicv3ConfigIsValid function

Returns TRUE if `ICH_LR<n>_EL2.HW == '0'` for all implemented values of `n`.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
readonly func Gicv3ConfigIsValid() => boolean
```

See also:

- A6.1 *Realm interrupts*
- B4.6.64 *RmiRecEnter type*

B3.32 GptL0Walk function

Returns the Level 1 GPT which describes the physical address region starting from `addr`.

```
readonly func GptL0Walk(
    addr : Address) => RmmGptL0Entry
```

B3.33 GptL1IsHomogeneous function

Returns TRUE if the Level 1 GPT which describes the physical address region starting from `addr` is homogeneous.

```
readonly func GptL1IsHomogeneous(
    addr : Address) => boolean
```

B3.34 GranuleAt function

Returns the Granule located at physical address `addr`.

```
readonly func GranuleAt(
    addr : Address) => RmmGranule
```

See also:

- A2.3 *Physical memory*

DRAFT

B3.35 GranulesAllState function

Returns TRUE if the state of all Granules within the range `[base, top)` or `[base, base + size)` is equal to `state`.

```
readonly func GranulesAllState(
    base : Address,
    top : Address,
    state : RmmGranuleState) => boolean
```

```
readonly func GranulesAllState(
    base : Address,
    size : integer,
    state : RmmGranuleState) => boolean
```

B3.36 GranulesAllStateList function

Returns TRUE if the state of all Granules within the first `size` bytes of the address set described by an RMI Address Range List stored at PA `alist` is equal to `state`.

```
readonly func GranulesAllStateList(
    alist : Address,
    size : integer,
    state : RmmGranuleState) => boolean
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.37 GranuleSizeFromRmi function

Decodes a Granule size.

```
readonly func GranuleSizeFromRmi(
    size : RmiGranuleSize) => integer
begin
    case size of
        when RMI_GRANULE_SIZE_4KB  => return  4 * KB;
        when RMI_GRANULE_SIZE_16KB => return 16 * KB;
        when RMI_GRANULE_SIZE_64KB => return 64 * KB;
    end;
end;
```

B3.38 GranuleSizeToRmi function

Encodes a Granule size.

```
readonly func GranuleSizeToRmi(
    size : integer) => RmiGranuleSize
begin
    case size of
        when  4 * KB => return RMI_GRANULE_SIZE_4KB;
        when 16 * KB => return RMI_GRANULE_SIZE_16KB;
        when 64 * KB => return RMI_GRANULE_SIZE_64KB;
    end;
end;
```

B3.39 MecidAvailable function

Returns TRUE if a MEC is available which satisfies `policy`.

```
readonly func MecidAvailable(
    policy : RmiMecPolicy) => boolean
```

See also:

- Chapter A11 *Realm memory encryption*

B3.40 MemCategoryIsCompatible function

Returns TRUE if the specified memory category is compatible with the system memory layout view for the specified address.

```
readonly impdef func MemCategoryIsCompatible(
    category : RmiMemCategory,
    addr : Address) => boolean
```

See also:

- A2.3.2 *Views of physical memory*

B3.41 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.

```
readonly impdef func MemPermLabelSupported(
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    label : bits(64)) => boolean
```

B3.42 MinAddress function

Returns the smaller of two addresses.

```
readonly func MinAddress(
    addr1 : Address,
    addr2 : Address) => Address
begin
    return ToAddress(Min(UInt(addr1), UInt(addr2)));
end;
```

B3.43 MpidrEqual function

Returns TRUE if the specified MPIDR values are logically equivalent.

```
pure 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;
```

B3.44 MpidrIsUsed function

Returns TRUE if the specified MPIDR value identifies a REC in the current Realm.

```
readonly func MpidrIsUsed(
    mpidr : bits(64)) => boolean
```

B3.45 MsiAddrIsValid function

Returns TRUE if `addr` is a valid MSI address.

```
readonly func MsiAddrIsValid(
    addr : Address) => boolean
```

B3.46 NonSecureAccessPermitted function

Returns TRUE if the Granule located at physical address `addr` is accessible via Non-secure PAS.

```
readonly func NonSecureAccessPermitted(
    addr : Address) => boolean
```

B3.47 OperationCanCancel function

Returns whether the operation identified by `handle` can be cancelled.

```
readonly impdef func OperationCanCancel(
    handle : bits(64)) => RmmOpCanCancel
```

See also:

- B4.3.2 *Stateful RMI operations*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.48 OperationIncomplete function

Returns TRUE if the operation identified by `handle` results in an intermediate state.

```
readonly impdef func OperationIncomplete(
    handle : bits(64)) => boolean
```

See also:

- B4.3.2 *Stateful RMI operations*

B3.49 PaAllDelegableConventional16 function

Returns TRUE if the first `count` entries in a list addresses are delegable conventional memory.

```
readonly func PaAllDelegableConventional16(
    addr : array [[16]] of Address,
    count : integer) => boolean
begin
    assert 0 <= count && count <= 16;
    for i = 0 to count - 1 do
        if !PaIsDelegableConventionalFine(addr[[i]]) then
            return FALSE;
        end;
    end;
    return TRUE;
end;
```

B3.50 PaAllDelegableConventional32 function

Returns TRUE if the first `count` entries in a list addresses are delegable conventional memory.

```
readonly func PaAllDelegableConventional32(
    addr : array [[32]] of Address,
    count : integer) => boolean
begin
    assert 0 <= count && count <= 32;
    for i = 0 to count - 1 do
        if !PaIsDelegableConventionalFine(addr[[i]]) then
            return FALSE;
        end;
    end;
    return TRUE;
end;
```

B3.51 PaIsCohDevMem function

Returns TRUE if the Granule located at physical address `addr` is within a region of the system memory map which is reserved for coherent device memory.

Note that a TRUE return value does not indicate whether the RMM has verified a device which is mapped at this address.

```
readonly impdef func PaIsCohDevMem(
    addr : Address) => boolean
```

See also:

- A2.3.2 *Views of physical memory*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.52 PaIsDelegable function

Returns TRUE if the Granule located at physical address `addr` is delegable memory.

```
readonly func PaIsDelegable(
    addr : Address) => boolean
begin
    return (PaIsDelegableDevMem(addr)
        || PaIsDelegableConventionalFine(addr));
end;
```

See also:

- A2.3.5 *Delegable physical memory*

B3.53 PaIsDelegableCohDevMem function

Returns TRUE if the Granule located at physical address `addr` is delegable coherent device memory.

```
readonly func PaIsDelegableCohDevMem(
    addr : Address) => boolean
begin
    return (PaIsCohDevMem(addr)
        && PaIsPopulated(addr)
        && PaIsTracked(addr));
end;
```

See also:

- A2.3.5 *Delegable physical memory*

B3.54 PaIsDelegableConventional function

Returns TRUE if the Granule located at physical address `addr` is delegable conventional memory.

```
readonly func PaIsDelegableConventional(
    addr : Address) => boolean
begin
    var region : RmmTrackingRegion = TrackingRegionAt(addr);

    return (PaIsDram(addr)
            && PaIsTracked(addr))
        || (PaIsDelegableCohDevMem(addr)
            && region.category == MEM_CATEGORY_CONVENTIONAL);
end;
```

See also:

- A2.3.5 *Delegable physical memory*

B3.55 PaIsDelegableConventionalFine function

Returns TRUE if the Granule located at physical address `addr` is delegable conventional memory and is fine-grained tracked.

```
readonly func PaIsDelegableConventionalFine(
    addr : Address) => boolean
begin
    var region : RmmTrackingRegion = TrackingRegionAt(addr);
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    return (PaIsDelegableConventional(addr)
        && (region.state == TRACKING_FINE));
end;
```

B3.56 PaIsDelegableDevMem function

Returns TRUE if the Granule located at physical address `addr` is delegable device memory.

```
readonly func PaIsDelegableDevMem(
    addr : Address) => boolean
begin
    return (PaIsDelegableCohDevMem(addr)
        || PaIsDelegableNonCohDevMem(addr));
end;
```

See also:

- A2.3.5 *Delegable physical memory*

B3.57 PaIsDelegableNonCohDevMem function

Returns TRUE if the Granule located at physical address `addr` is delegable non-coherent device memory.

```
readonly func PaIsDelegableNonCohDevMem(
    addr : Address) => boolean
begin
    return (PaIsNonCohDevMem(addr)
        && PaIsPopulated(addr)
        && PaIsTracked(addr));
end;
```

See also:

- A2.3.5 *Delegable physical memory*

B3.58 PaIsDram function

Returns TRUE if the Granule located at physical address `addr` is within a region of the system memory map which is backed by DRAM.

```
readonly impdef func PaIsDram(
    addr : Address) => boolean
```

See also:

- A2.3.2 *Views of physical memory*

B3.59 PaIsNonCohDevMem function

Returns TRUE if the Granule located at physical address `addr` is within a region of the system memory map which is reserved for non-coherent device memory.

Note that a TRUE return value does not indicate whether the RMM has verified a device which is mapped at this address.

```
readonly impdef func PaIsNonCohDevMem(
    addr : Address) => boolean
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- A2.3.2 *Views of physical memory*

B3.60 PaIsPopulated function

Returns TRUE if physical address `addr` is backed by a resource which has been verified by the RMM.

```
readonly func PaIsPopulated(
    addr : Address) => boolean
```

See also:

- A2.3.3 *Populated physical memory*
- B3.61 *PaIsTracked function*

B3.61 PaIsTracked function

Returns TRUE if the tracking region which includes address `addr` is tracked.

Note that a TRUE return value does not indicate whether the address is backed by a resource which has been verified by the RMM.

```
readonly func PaIsTracked(
    addr : Address) => boolean
begin
    var region : RmmTrackingRegion = TrackingRegionAt(addr);

    return (region.state == TRACKING_COARSE
        || region.state == TRACKING_FINE);
end;
```

See also:

- A2.3.4 *Granule tracking region*
- B3.60 *PaIsPopulated function*

B3.62 PaIsTrackedFine function

Returns TRUE if the tracking region which includes address `addr` is tracked at fine granularity.

Note that a TRUE return value does not indicate whether the address is backed by a resource which has been verified by the RMM.

```
readonly func PaIsTrackedFine(
    addr : Address) => boolean
begin
    var region : RmmTrackingRegion = TrackingRegionAt(addr);

    return region.state == TRACKING_FINE;
end;
```

B3.63 PaRangeIsPopulated function

Returns TRUE if all physical addresses within the range `[base, top)` are backed by resources which have been verified by the RMM.

```
readonly func PaRangeIsPopulated(
    base : Address,
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    top : Address) => boolean
```

See also:

- A2.3.3 *Populated physical memory*

B3.64 PaRangeIsUnpopulated function

Returns TRUE if no physical addresses within the range `[base, top)` are backed by resources which have been verified by the RMM.

```
readonly func PaRangeIsUnpopulated(
    base : Address,
    top : Address) => boolean
```

See also:

- A2.3.3 *Populated physical memory*

B3.65 PdevAt function

Returns the PDEV object located at physical address `addr`.

```
readonly func PdevAt(
    addr : Address) => RmmPdev
```

B3.66 PdevFlags function

Get RmiPdevFlags value.

```
readonly func PdevFlags(
    pdev : RmmPdev) => RmiPdevFlags
begin
    var flags : RmiPdevFlags;

    case pdev.spdm of
        when SPDM_FALSE => flags.spdm = RMI_SPDM_FALSE;
        when SPDM_TRUE => flags.spdm = RMI_SPDM_TRUE;
    end;

    return flags;
end;
```

B3.67 PdevIsBusy function

Returns TRUE if the PDEV object is currently unable to service a request for an IMPLEMENTATION DEFINED reason.

```
readonly impdef func PdevIsBusy(
    pdev : RmmPdev) => boolean
```

B3.68 PdevRidRangeOverlap function

Returns TRUE if the RID range of `pdev` overlaps the RID range of any other protected PDEV within the same PCIe segment.

```
readonly impdef func PdevRidRangeOverlap(
    pdev : RmmPdev) => boolean
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.69 PdevStreamAlloc function

Allocate a PDEV stream handle for the specified pair of PDEVs.

If a free handle is available, result.valid is TRUE and result.stream.state is PDEV_STREAM_DISCONNECTED.

```
readonly func PdevStreamAlloc(
    pdev_1 : RmmPdev,
    pdev_2 : RmmPdev) => RmmPdevStreamResult
```

B3.70 PdevStreamFromHandle function

Returns PDEV stream.

```
readonly func PdevStreamFromHandle(
    pdev : RmmPdev,
    stream_hnd : bits(64)) => RmmPdevStreamResult
```

```
readonly func PdevStreamFromHandle(
    pdev_1 : RmmPdev,
    pdev_2 : RmmPdev,
    stream_hnd : bits(64)) => RmmPdevStreamResult
```

B3.71 PdevStreamFromType function

Query whether the specified PDEV(s) have a stream of type `stream_type`.

```
readonly func PdevStreamFromType(
    pdev : RmmPdev,
    stream_type : RmmPdevStreamType) => RmmPdevStreamResult
```

```
readonly func PdevStreamFromType(
    pdev_1 : RmmPdev,
    pdev_2 : RmmPdev,
    stream_type : RmmPdevStreamType) => RmmPdevStreamResult
```

B3.72 PdevStreamLive function

Returns TRUE if `pdev` has any connected PDEV streams.

```
readonly func PdevStreamLive(
    pdev : RmmPdev) => boolean
```

B3.73 PdevStreamPdev2Category function

Returns category of pdev_2 for the specified PDEV stream type.

```
readonly func PdevStreamPdev2Category(
    stream_type : RmiPdevStreamType) => RmmPdevCategory
begin
    case stream_type of
        when RMI_PDEV_STREAM_NON_TEE  => return PDEV_ROOT_PORT;
        when RMI_PDEV_STREAM_NCOH     => return PDEV_ROOT_PORT;
        when RMI_PDEV_STREAM_COH      => return PDEV_ROOT_PORT;
        otherwise                     => unreachable;
    end;
end;
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.74 PdevStreamPdev2Required function

Returns TRUE if pdev_2 is required for the specified PDEV stream type.

```
readonly func PdevStreamPdev2Required(
    stream_type : RmmPdevStreamType) => boolean
begin
    case stream_type of
        when PDEV_STREAM_NCOH_SYS => return FALSE;
        when PDEV_STREAM_COH_SYS  => return FALSE;
        otherwise                 => return TRUE;
    end;
end;
```

```
readonly func PdevStreamPdev2Required(
    stream_type : RmiPdevStreamType) => boolean
begin
    return PdevStreamPdev2Required(
        PdevStreamTypeFromRmi(stream_type));
end;
```

B3.75 PdevStreamsForVdev function

Returns TRUE if `pdev` has the required set of streams for VDEV creation.

```
readonly func PdevStreamsForVdev(
    pdev : RmmPdev) => boolean
```

See also:

- A9.3 *Physical device stream object*

B3.76 PdevStreamTypeFromRmi function

Convert stream type enumeration.

```
readonly func PdevStreamTypeFromRmi(
    stream_type : RmiPdevStreamType) => RmmPdevStreamType
begin
    case stream_type of
        when RMI_PDEV_STREAM_NON_TEE  => return PDEV_STREAM_NON_TEE;
        when RMI_PDEV_STREAM_NCOH     => return PDEV_STREAM_NCOH;
        when RMI_PDEV_STREAM_COH      => return PDEV_STREAM_COH;
        when RMI_PDEV_STREAM_NCOH_SYS => return PDEV_STREAM_NCOH_SYS;
        when RMI_PDEV_STREAM_COH_SYS  => return PDEV_STREAM_COH_SYS;
    end;
end;
```

B3.77 PdevVsmmuIsCompatible 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`.

```
readonly func PdevVsmmuIsCompatible(
    pdev : RmmPdev,
    vsmmu : RmmVsmmu) => boolean
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.78 PlaneSysregValid function

Whether `addr` identifies an accessible Plane system register.

If `addr.d128` is TRUE and either of the following is false then the function returns FALSE.

- FEAT_SYSREG128 is implemented
- `addr` identifies a 128-bit system register

```
readonly func PlaneSysregValid(
    rec : RmmRec,
    addr : RsiSysregAddress,
    op : RmmReadWriteOp) => boolean
```

B3.79 PlaneSysregValue function

Value of a Plane system register.

```
readonly func PlaneSysregValue(
    rec : RmmRec,
    plane_idx : integer,
    addr : RsiSysregAddress) => bits(128)
```

B3.80 PsciReturnCodeEncode function

Return encoding for a PsciReturnCode value.

```
pure func PsciReturnCodeEncode(
    value : PsciReturnCode) => bits(64)
```

B3.81 PsciReturnCodePermitted function

Whether a PSCI return code is permitted.

```
readonly 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*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.28 *RMI_PSCI_COMPLETE command*

B3.82 PsciVersion function

PSCI version.

```
readonly func PsciVersion() => PsciInterfaceVersion
```

B3.83 PsmmuAddrIsValid function

Returns TRUE if `addr` is the base address of a PSMMU.

```
readonly func PsmmuAddrIsValid(
    addr : Address) => boolean
```

B3.84 PsmmuAt function

Returns the PSMMU object located at physical address `addr`.

```
readonly func PsmmuAt(
    addr : Address) => RmmPsmmu
```

B3.85 PsmmuCmdQueueFull function

Returns `TRUE` if the command queue of `psmmu` is full.

```
readonly func PsmmuCmdQueueFull(
    psmmu : RmmPsmmu) => boolean
```

B3.86 PsmmuFromPdev function

Returns the PSMMU which handles transactions from `pdev`.

```
readonly func PsmmuFromPdev(
    pdev : RmmPdev) => RmmPsmmu
```

B3.87 PsmmuInfoAt function

Returns the RmiPsmmuInfo object located at physical address `addr`.

```
readonly func PsmmuInfoAt(
    addr : Address) => RmiPsmmuInfo
```

B3.88 PsmmuL1StIsLive function

Returns TRUE if the PSMMU Level 1 Stream Table is live.

```
readonly func PsmmuL1StIsLive(
    psmmu : RmmPsmmu) => boolean
```

See also:

- A9.7.4 *PSMMU Stream Tables*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.89 PsmmuL2StIsLive function

Returns TRUE if the PSMMU Level 2 Stream Table which describes StreamID range starting at `sid` is live.

```
readonly func PsmmuL2StIsLive(
    psmmu : RmmPsmmu,
    sid : bits(64)) => boolean
```

See also:

- A9.7.4 *PSMMU Stream Tables*

B3.90 PsmmuStWalk function

Returns PSMMU Stream Table walk result for StreamID `sid`.

```
readonly func PsmmuStWalk(
    psmmu : RmmPsmmu,
    sid : bits(64)) => RmmPsmmuStWalkResult
```

B3.91 PsmmuSupportsMsi function

Returns TRUE if the PSMMU located at `addr` supports MSI.

```
readonly func PsmmuSupportsMsi(
    addr : Address) => boolean
```

B3.92 RealmAt function

Returns the Realm whose RD is located at physical address `addr`.

```
readonly func RealmAt(
    addr : Address) => RmmRealm
```

See also:

- A2.2 *Realm*

B3.93 RealmIpaRangeAllRipasIf function

Returns TRUE if all addresses within IPA range [base, top) for which RIPAS was equal to `ripas_pre` in `realm_pre` have RIPAS equal to `ripas` in `realm`.

```
readonly func RealmIpaRangeAllRipasIf(
    realm_pre : RmmRealm,
    realm : RmmRealm,
    base : Address,
    top : Address,
    ripas_pre : RmmRipas,
    ripas : RmmRipas) => boolean
```

B3.94 RealmIsLive function

Returns TRUE if the Realm whose RD is located at physical address `addr` is live.

```
readonly func RealmIsLive(
    addr : Address) => boolean
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- A2.2.4 *Realm liveness*

B3.95 RealmMeasurementEncode function

Return encoding for an RmmRealmMeasurement value.

```
pure func RealmMeasurementEncode(
    value : RmmRealmMeasurement) => array [[8]] of bits(64)
```

B3.96 RealmParamsSupported function

Returns TRUE if the Realm parameters are supported by the implementation.

```
readonly func RealmParamsSupported(
    params : RmiRealmParams) => boolean
begin
    var rmm : RmmGlobal = Rmm();

    if (params.flags0.lpa2 == RMI_FEATURE_TRUE
        && rmm.static.feat_lpa2 != FEATURE_TRUE) then
        return FALSE;
    end;

    if (params.flags0.sve == RMI_FEATURE_TRUE
        && rmm.static.feat_sve != FEATURE_TRUE) then
        return FALSE;
    end;

    if (params.flags0.pmu == RMI_FEATURE_TRUE
        && rmm.static.feat_pmu != FEATURE_TRUE) then
        return FALSE;
    end;

    if (params.flags0.da == RMI_FEATURE_TRUE
        && rmm.static.feat_da != FEATURE_TRUE) then
        return FALSE;
    end;

    if (params.flags1.ats == RMI_FEATURE_TRUE
        && rmm.static.feat_ats != FEATURE_TRUE) then
        return FALSE;
    end;

    if (params.s2sz > rmm.static.max_ipa_width) then
        return FALSE;
    end;

    if (params.sve_vl > rmm.static.max_sve_vl) then
        return FALSE;
    end;

    if (params.num_bps == 0
        || params.num_bps + 1 > rmm.static.num_bps) then
        return FALSE;
    end;

    if (params.num_wps == 0
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            || params.num_wps + 1 > rmm.static.num_wps) then
            return FALSE;
        end;

        if (params.pmu_num_ctrs > rmm.static.pmu_num_ctrs) then
            return FALSE;
        end;

        if (params.hash_algo == RMI_HASH_SHA_256
            && rmm.static.feat_sha_256 != FEATURE_TRUE) then
            return FALSE;
        end;

        if (params.hash_algo == RMI_HASH_SHA_384
            && rmm.static.feat_sha_384 != FEATURE_TRUE) then
            return FALSE;
        end;

        if (params.hash_algo == RMI_HASH_SHA_512
            && rmm.static.feat_sha_512 != FEATURE_TRUE) then
            return FALSE;
        end;

        if (params.num_aux_planes > rmm.static.max_num_aux_planes) then
            return FALSE;
        end;

        if (params.flags1.rtt_s2ap_encoding == RMI_S2AP_INDIRECT
            && rmm.static.rtt_s2ap_indirect == FEATURE_FALSE) then
            return FALSE;
        end;

        if (params.num_aux_planes > 0) then
            if (params.flags1.rtt_tree_per_plane == RMI_FEATURE_FALSE
                && rmm.static.rtt_plane == RTT_PLANE_AUX) then
                return FALSE;
            end;

            if (params.flags1.rtt_tree_per_plane == RMI_FEATURE_TRUE
                && rmm.static.rtt_plane == RTT_PLANE_SINGLE) 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:

- A2.2.6 *Realm parameters*
- Chapter A3 *Feature discovery and configuration*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.97 RealmRttBaseEqual function

Returns TRUE if RTT base values of `realm` match the provided values.

```
pure 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;
```

B3.98 RecAt function

Returns the REC object located at physical address `addr`.

```
readonly func RecAt(
    addr : Address) => RmmRec
```

See also:

- A2.4 *Realm Execution Context*

B3.99 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.

```
readonly impdef func RecAuxCount(
    rd : Address) => integer
```

B3.100 RecDevMemResponseToRsi function

Returns response to VDEV mapping validation request.

```
readonly func RecDevMemResponseToRsi(
    rec : RmmRec) => RsiResponse
begin
    if ((rec.dev_mem_addr != rec.dev_mem_top)
        && (rec.dev_mem_response == RESPONSE_REJECT)) then
        return RSI_RESPONSE_REJECT;
    end;

    return RSI_RESPONSE_ACCEPT;
end;
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.101 RecFromMpidr function

Returns the REC object identified by the specified MPIDR value, in the current Realm.

```
readonly func RecFromMpidr(
    mpidr : bits(64)) => RmmRec
```

B3.102 RecIndex function

Returns the REC index which corresponds to `mpidr`.

```
pure 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:

- A2.4.3 *REC index and MPIDR value*

B3.103 RecRipasResponseToRsi function

Returns response to RIPAS change request.

```
readonly func RecRipasResponseToRsi(
    rec : RmmRec) => RsiResponse
begin
    if ((rec.ripas_value == RIPAS_RAM)
            && (rec.ripas_addr != rec.ripas_top)
            && (rec.ripas_response == RESPONSE_REJECT)) then
        return RSI_RESPONSE_REJECT;
    end;

    return RSI_RESPONSE_ACCEPT;
end;
```

See also:

- A5.4 *RIPAS change*

B3.104 RecS2APResponseToRsi function

Returns response to S2AP change request.

```
readonly func RecS2APResponseToRsi(
    rec : RmmRec) => RsiResponse
begin
    if ((rec.s2ap_addr != rec.s2ap_top)
            && (rec.s2ap_response == RESPONSE_REJECT)) then
        return RSI_RESPONSE_REJECT;
    end;

    return RSI_RESPONSE_ACCEPT;
end;
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- A10.3.2.3 *Stage 2 Access Permissions change within a multi-Plane Realm*

B3.105 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`.

```
readonly func RemExtend(
    hash_algo : RmmHashAlgorithm,
    old_value : RmmRealmMeasurement,
    new_value : RmmRealmMeasurement,
    size : integer) => RmmRealmMeasurement
```

See also:

- A7.1.2 *Realm Extensible Measurement*

B3.106 RimExtendData function

Extend RIM with contribution from DATA creation.

```
readonly func RimExtendData(
    realm : RmmRealm,
    ipa : Address,
    data : Address,
    flags : RmiDataFlags) => RmmRealmMeasurement
```

See also:

- B4.5.58.4 *RMI_RTT_DATA_MAP_INIT extension of RIM*

B3.107 RimExtendRec function

Extend RIM with contribution from REC creation.

```
readonly func RimExtendRec(
    realm : RmmRealm,
    params : RmiRecParams) => RmmRealmMeasurement
```

See also:

- B4.5.40.4 *RMI_REC_CREATE extension of RIM*

B3.108 RipasToRmi function

Encodes a RIPAS value.

```
readonly func RipasToRmi(
    ripas : RmmRipas) => RmiRipas
begin
    case ripas of
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        when RIPAS_EMPTY     => return RMI_RIPAS_EMPTY;
        when RIPAS_RAM       => return RMI_RIPAS_RAM;
        when RIPAS_DESTROYED => return RMI_RIPAS_DESTROYED;
        when RIPAS_DEV       => return RMI_RIPAS_DEV;
    end;
end;
```

B3.109 RmiAddrBlockSizeToLevel function

Decode an RMI Address Block Size.

```
readonly func RmiAddrBlockSizeToLevel(
    size : RmiAddrBlockSize) => integer
begin
    case size of
        when RMI_BLOCK_L0 => return 0;
        when RMI_BLOCK_L1 => return 1;
        when RMI_BLOCK_L2 => return 2;
        when RMI_PAGE_L3  => return 3;
        otherwise => unreachable;
    end;
end;
```

B3.110 RmiAddrRangeDescDecode function

Decode an RMI Address Range Descriptor.

```
readonly func RmiAddrRangeDescDecode(
    desc : RmiAddrRangeDesc) => RmmAddrRange
begin
    var rmm : RmmGlobal = Rmm();

    var base : integer;
    var count : integer;
    var level : integer;

    case rmm.dynamic.rmi_granule_size of
        when 4 * KB =>
            base = UInt(desc.data.granule_4kb.addr);
            count = desc.data.granule_4kb.count;
            level = RmiAddrBlockSizeToLevel(
                desc.data.granule_4kb.size);

        when 16 * KB =>
            base = UInt(desc.data.granule_16kb.addr);
            count = desc.data.granule_16kb.count;
            level = RmiAddrBlockSizeToLevel(
                desc.data.granule_16kb.size);

        when 64 * KB =>
            base = UInt(desc.data.granule_64kb.addr);
            count = desc.data.granule_64kb.count;
            level = RmiAddrBlockSizeToLevel(
                desc.data.granule_64kb.size);

        otherwise =>
            unreachable;
    end;
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    return RmmAddrRange {
        base = ToAddress(base),
        top = ToAddress(base + count * RttLevelSize(level))
    };
end;
```

B3.111 RmiAddrRangesEqual function

Returns TRUE if two address ranges are equal.

```
pure func RmiAddrRangesEqual(
    range1 : RmmAddrRange,
    range2 : RmiAddrRange) => boolean
begin
    if range1.base != range2.base then
        return FALSE;
    end;
    if range1.top != range2.top then
        return FALSE;
    end;
    return TRUE;
end;
```

B3.112 RmiAddrRangesEqual16 function

Returns TRUE if the first `count` entries in two arrays of address ranges are equal.

```
pure func RmiAddrRangesEqual16(
    ranges1 : array [[16]] of RmmAddrRange,
    ranges2 : array [[16]] of RmiAddrRange,
    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;
```

B3.113 RmiAddrRangesEqual8 function

Returns TRUE if the first `count` entries in two arrays of address ranges are equal.

```
pure func RmiAddrRangesEqual8(
    ranges1 : array [[8]] of RmmAddrRange,
    ranges2 : array [[8]] of RmiAddrRange,
    count : integer) => boolean
begin
    assert 0 <= count && count <= 8;
    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
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            return FALSE;
        end;
    end;
    return TRUE;
end;
```

B3.114 RmiAddrRangesValid8 function

Returns TRUE if address ranges are sorted and non-overlapping.

```
readonly func RmiAddrRangesValid8(
    addr_range : array [[8]] of RmiAddrRange,
    num_addr_range : integer) => boolean
begin
    var last_top : Address = ARBITRARY : Address;

    for i = 0 to num_addr_range - 1 do
        var range : RmiAddrRange = addr_range[[i]];
        if UInt(range.base) >= UInt(range.top) then
            return FALSE;
        end;
        if i > 0 && UInt(range.base) < UInt(last_top) then
            return FALSE;
        end;
        last_top = range.top;
    end;

    return TRUE;
end;
```

B3.115 RmiDevCommComplete function

Device communication is complete.

```
readonly func RmiDevCommComplete(
    flags : RmiDevCommExitFlags) => boolean
begin
    return (
        flags.req_cache == RMI_FALSE
        && flags.rsp_cache == RMI_FALSE
        && flags.req_send == RMI_FALSE
        && flags.rsp_wait == RMI_FALSE
        && flags.rsp_reset == RMI_FALSE);
end;
```

B3.116 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.

```
readonly func RmiDevCommDataAt(
    addr : Address) => RmiDevCommData
```

B3.117 RmiFeatureRegister0Decode function

Decode RmiFeatureRegister0 value.

```
readonly func RmiFeatureRegister0Decode(
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    value : bits(64)) => RmiFeatureRegister0
```

B3.118 RmiFeatureRegisterEncode function

Encode feature register.

```
readonly func RmiFeatureRegisterEncode(
    index : integer) => bits(64)
begin
    var rmm : RmmGlobal = Rmm();
    var result : bits(64) = Zeros{64}();

    if (index == 0) then
        var reg : RmiFeatureRegister0;

        reg.S2SZ = rmm.static.max_ipa_width;
        reg.LPA2 = FeatureToRmi(rmm.static.feat_lpa2);
        reg.SVE = FeatureToRmi(rmm.static.feat_sve);
        reg.SVE_VL = rmm.static.max_sve_vl;

        assert rmm.static.num_bps >= 2 && rmm.static.num_bps <= 2^6;
        reg.NUM_BPS = rmm.static.num_bps - 1;

        assert rmm.static.num_wps >= 2 && rmm.static.num_wps <= 2^6;
        reg.NUM_WPS = rmm.static.num_wps - 1;

        reg.PMU = FeatureToRmi(rmm.static.feat_pmu);
        reg.PMU_NUM_CTRS = rmm.static.pmu_num_ctrs;

        // Omitted: encode reg into bits(64) value
    end;

    if (index == 1) then
        var reg : RmiFeatureRegister1;

        reg.RMI_GRAN_SZ_4KB = FeatureToRmi(rmm.static.rmi_granule_size_4kb);
        reg.RMI_GRAN_SZ_16KB = FeatureToRmi(rmm.static.rmi_granule_size_16kb);
        reg.RMI_GRAN_SZ_64KB = FeatureToRmi(rmm.static.rmi_granule_size_64kb);

        reg.HASH_SHA_256 = FeatureToRmi(rmm.static.feat_sha_256);
        reg.HASH_SHA_384 = FeatureToRmi(rmm.static.feat_sha_384);
        reg.HASH_SHA_512 = FeatureToRmi(rmm.static.feat_sha_512);

        reg.MAX_RECS_ORDER = rmm.static.max_recs_order;

        // Omitted: set reg.L0GPTSZ
        // Omitted: set reg.PPS

        // Omitted: encode reg into bits(64) value
    end;

    if (index == 2) then
        var reg : RmiFeatureRegister2;

        reg.DA = FeatureToRmi(rmm.static.feat_da);
        reg.DA_COH = FeatureToRmi(rmm.static.feat_da_coh);
        reg.VSMMU = FeatureToRmi(rmm.static.feat_vsmmu);
        reg.ATS = FeatureToRmi(rmm.static.feat_ats);
        reg.NON_TEE_STREAM = FeatureToRmi(rmm.static.feat_non_tee_stream);
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        reg.MAX_VDEVS_ORDER = rmm.static.max_vdevs_order;
        reg.VDEV_KROU = FeatureToRmi(rmm.static.feat_vdev_krou);

        // Omitted: encode reg into bits(64) value
    end;

    if (index == 3) then
        var reg : RmiFeatureRegister3;

        reg.MAX_NUM_AUX_PLANES = rmm.static.max_num_aux_planes;

        case rmm.static.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(rmm.static.rtt_s2ap_indirect);

        // Omitted: encode reg into bits(64) value
    end;

    if (index == 4) then
        var reg : RmiFeatureRegister4;

        reg.MEC_COUNT = rmm.static.mec_count;

        // Omitted: encode reg into bits(64) value
    end;

    return result;
end;
```

B3.119 RmiPdevFlagsDecode function

Decode RmiPdevFlags value.

```
readonly func RmiPdevFlagsDecode(
    value : bits(64)) => RmiPdevFlags
```

B3.120 RmiPdevFlagsSupported function

Returns TRUE if the provided PDEV flags are supported by the implementation.

```
readonly func RmiPdevFlagsSupported(
    flags : RmiPdevFlags) => boolean
begin
    var rmm : RmmGlobal = Rmm();

    if (rmm.static.feat_da != FEATURE_TRUE) then
        return FALSE;
    end;

    // Omitted: IMPDEF check for whether the PDEV category is
    // supported.

    // Omitted: IMPDEF checks for whether the following flags
    // are supported:
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    // - spdm

    return TRUE;
end;
```

See also:

- A9.2.1 *Physical device attributes*

B3.121 RmiPdevParamsAt function

Returns PDEV parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
readonly func RmiPdevParamsAt(
    addr : Address) => RmiPdevParams
```

B3.122 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

```
readonly func RmiPdevParamsIsValid(
    addr : Address) => boolean
```

DRAFT

B3.123 RmiPdevStreamParamsAt function

Returns PDEV stream parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
readonly func RmiPdevStreamParamsAt(
    addr : Address) => RmiPdevStreamParams
```

B3.124 RmiPdevStreamParamsIsValid function

Returns TRUE if the memory location contains a valid encoding of the RmiPdevStreamParams type and all the following are true:

- The IDE stream identifier is valid
- 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 either PDEV
- None of the address ranges overlaps an address range for another PDEV
- A connection between the specified PDEV(s) is supported by the implementation

```
readonly func RmiPdevStreamParamsIsValid(
    addr : Address) => boolean
```

B3.125 RmiPsmmuParamsAt function

Returns PSMMU parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
readonly func RmiPsmmuParamsAt(
    addr : Address) => RmiPsmmuParams
```

B3.126 RmiPublicKeyParamsAt function

Returns public key parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
readonly func RmiPublicKeyParamsAt(
    addr : Address) => RmiPublicKeyParams
```

B3.127 RmiRealmParamsAt function

Returns Realm parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
readonly func RmiRealmParamsAt(
    addr : Address) => RmiRealmParams
```

See also:

- A2.2.6 *Realm parameters*

B3.128 RmiRealmParamsIsValid function

Returns TRUE if the memory location contains a valid encoding of the RmiRealmParams type.

```
readonly func RmiRealmParamsIsValid(
    addr : Address) => boolean
```

DRAFT

B3.129 RmiRecParamsAt function

Returns REC parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
readonly func RmiRecParamsAt(
    addr : Address) => RmiRecParams
```

B3.130 RmiRecRunAt function

Returns the RecRun object stored at physical address `addr`.

```
readonly func RmiRecRunAt(
    addr : Address) => RmiRecRun
```

See also:

- A4.2 *REC entry*
- A4.3 *REC exit*

B3.131 RmiRmmConfigAt function

Returns system configuration stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
readonly func RmiRmmConfigAt(
    addr : Address) => RmiRmmConfig
```

B3.132 RmiVdevFlagsDecode function

Decode RmiVdevFlags value.

```
pure func RmiVdevFlagsDecode(
    value : bits(64)) => RmiVdevFlags
```

B3.133 RmiVdevMeasureParamsAt function

Returns device measurement parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
readonly func RmiVdevMeasureParamsAt(
    addr : Address) => RmiVdevMeasureParams
```

B3.134 RmiVdevParamsAt function

Returns VDEV parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
readonly func RmiVdevParamsAt(
    addr : Address) => RmiVdevParams
```

B3.135 RmiVdevParamsIsValid function

Returns TRUE if the memory location contains a valid encoding of the RmiPdevParams type.

```
readonly func RmiVdevParamsIsValid(
    addr : Address) => boolean
```

B3.136 RmiVersionHigherIsSupported function

Returns TRUE if the RMM supports an RMI revision which is incompatible with and greater than `version`.

```
readonly func RmiVersionHigherIsSupported(
    version : RmiInterfaceVersion) => boolean
```

See also:

- Chapter B2 *Interface versioning*
- B4.5.82 *RMI_VERSION command*

B3.137 RmiVersionHighest function

Returns the highest RMI revision supported by the RMM.

```
readonly func RmiVersionHighest() => RmiInterfaceVersion
```

See also:

- Chapter B2 *Interface versioning*
- B4.5.82 *RMI_VERSION command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.138 RmiVersionHighestBelow function

Returns the highest RMI revision which is both less than `version` and supported by the RMM.

```
readonly func RmiVersionHighestBelow(
    version : RmiInterfaceVersion) => RmiInterfaceVersion
```

See also:

- Chapter B2 *Interface versioning*
- B4.5.82 *RMI_VERSION command*

B3.139 RmiVersionIsSupported function

Returns TRUE if the RMM supports an RMI revision which is compatible with `version`.

```
readonly func RmiVersionIsSupported(
    version : RmiInterfaceVersion) => boolean
```

See also:

- Chapter B2 *Interface versioning*
- B4.5.82 *RMI_VERSION command*

B3.140 RmiVersionLowerIsSupported function

Returns TRUE if the RMM supports an RMI revision which is incompatible with and less than `version`.

```
readonly func RmiVersionLowerIsSupported(
    version : RmiInterfaceVersion) => boolean
```

See also:

- Chapter B2 *Interface versioning*
- B4.5.82 *RMI_VERSION command*

DRAFT

B3.141 RmiVsmmuParamsAt function

Returns VSMMU parameters stored at physical address `addr`.

If the PAS of `addr` is not NS, the return value is UNKNOWN.

```
readonly func RmiVsmmuParamsAt(
    addr : Address) => RmiVsmmuParams
```

See also:

- A9.8.3 *VSMMU liveness*

B3.142 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 set to an architecturally valid value.
- aidr value does not define any features which are unsupported by the RMM.
- All idr[] values are set to an architecturally valid value.
- No idr[] value defines any features which are unsupported by the RMM.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Here, “architecturally valid” refers to the SMMU implementation to which this VSMMU belongs.

```
readonly func RmiVsmmuParamsIsValid(
    addr : Address) => boolean
```

See also:

- *Arm System Memory Management Unit Architecture Specification* [22]
- A9.8 *Virtual SMMU*

B3.143 Rmm function

Returns global state of the implementation.

```
readonly impdef func Rmm() => RmmGlobal
```

See also:

- A2.1 *RMM*
- Chapter A3 *Feature discovery and configuration*

B3.144 RmmConfigIsSupported function

System configuration is supported.

```
readonly func RmmConfigIsSupported(
    cfg : RmiRmmConfig) => boolean
begin
    var rmm : RmmGlobal = Rmm();
    case cfg.rmi_granule_size of
        when RMI_GRANULE_SIZE_4KB =>
            if rmm.static.rmi_granule_size_4kb != FEATURE_TRUE then
                return FALSE;
            end;

        when RMI_GRANULE_SIZE_16KB =>
            if rmm.static.rmi_granule_size_16kb != FEATURE_TRUE then
                return FALSE;
            end;

        when RMI_GRANULE_SIZE_64KB =>
            if rmm.static.rmi_granule_size_64kb != FEATURE_TRUE then
                return FALSE;
            end;
    end;

    if (TrackingRegionSizeFromRmi(
            cfg.rmi_granule_size, cfg.tracking_region_size) == 0) then
        return FALSE;
    end;

    return TRUE;
end;
```

B3.145 RsiFeatureRegisterEncode function

Encode feature register.

```
readonly func RsiFeatureRegisterEncode(
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    realm : RmmRealm,
    index : integer) => bits(64)
begin
    var result : bits(64) = Zeros{64}();

    if (index == 0) then
        var reg : RsiFeatureRegister0;

        reg.DA = FeatureToRsi(realm.feat_da);

        // Omitted: set reg.MRO depending on whether platform
        // implements FEAT_S2PIE

        reg.ATS = FeatureToRsi(realm.feat_ats);

        // Omitted: encode reg into bits(64) value
    end;

    return result;
end;
```

B3.146 RsiHostCallAt function

Returns Host call data stored at IPA `addr`, mapped in the current Realm.

```
readonly func RsiHostCallAt(
    addr : Address) => RsiHostCall
```

B3.147 RsiPlaneRunAt function

Returns the PlaneRun object stored at IPA `addr`.

```
readonly func RsiPlaneRunAt(
    realm : RmmRealm,
    addr : Address) => RsiPlaneRun
```

B3.148 RsiRealmConfigAt function

Returns Realm configuration stored at IPA `addr`, mapped in the current Realm.

```
readonly func RsiRealmConfigAt(
    addr : Address) => RsiRealmConfig
```

B3.149 RsiRealmMeasurement function

Realm measurement value.

```
readonly func RsiRealmMeasurement(
    realm : RmmRealm,
    index : integer) => RmmRealmMeasurement
begin
    if (index == 0) then
        return realm.rim;
    end;

    assert 1 <= index && index <= 4;
    return realm.rem[[index - 1]];
end;
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.150 RsiVdevInfoAt function

Returns device configuration stored at IPA `addr`, mapped in the current Realm.

```
readonly func RsiVdevInfoAt(
    addr : Address) => RsiVdevInfo
```

B3.151 RsiVersionHigherIsSupported function

Returns TRUE if the RMM supports an RSI revision which is incompatible with and greater than `version`.

```
readonly func RsiVersionHigherIsSupported(
    version : RsiInterfaceVersion) => boolean
```

See also:

- Chapter B2 *Interface versioning*
- B5.4.21 *RSI_VERSION command*

B3.152 RsiVersionHighest function

Returns the highest RSI revision supported by the RMM.

```
readonly func RsiVersionHighest() => RsiInterfaceVersion
```

See also:

- Chapter B2 *Interface versioning*
- B5.4.21 *RSI_VERSION command*

B3.153 RsiVersionHighestBelow function

Returns the highest RSI revision which is both less than `version` and supported by the RMM.

```
readonly func RsiVersionHighestBelow(
    version : RsiInterfaceVersion) => RsiInterfaceVersion
```

See also:

- Chapter B2 *Interface versioning*
- B5.4.21 *RSI_VERSION command*

B3.154 RsiVersionIsSupported function

Returns TRUE if the RMM supports an RSI revision which is compatible with `version`.

```
readonly func RsiVersionIsSupported(
    version : RsiInterfaceVersion) => boolean
```

See also:

- Chapter B2 *Interface versioning*
- B5.4.21 *RSI_VERSION command*

B3.155 RsiVersionLowerIsSupported function

Returns TRUE if the RMM supports an RSI revision which is incompatible with and less than `version`.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
readonly func RsiVersionLowerIsSupported(
    version : RsiInterfaceVersion) => boolean
```

See also:

- Chapter B2 *Interface versioning*
- B5.4.21 *RSI_VERSION command*

B3.156 RttAllEntriesContiguous function

Returns TRUE if all entries in the RTT at address `rtt` at level `level` have contiguous output addresses, starting with `addr`.

```
readonly func RttAllEntriesContiguous(
    rtt : RmmRtt,
    addr : Address,
    level : integer) => boolean
```

See also:

- A5.6 *Realm Translation Table*

B3.157 RttAllEntriesMemAttr function

Returns TRUE if the memory attributes of all entries in the RTT at address `rtt` are equal to the memory attributes in `rtte`.

```
readonly func RttAllEntriesMemAttr(
    rtt : RmmRtt,
    rtte : RmmRttEntry) => boolean
```

B3.158 RttAllEntriesRipas function

Returns TRUE if all entries in the RTT at address `rtt` have RIPAS `ripas`.

```
readonly func RttAllEntriesRipas(
    rtt : RmmRtt,
    ripas : RmmRipas) => boolean
```

B3.159 RttAllEntriesS2AP function

Returns TRUE if the S2AP of all entries in the RTT at address `rtt` are equal to the S2AP in `rtte`.

```
readonly func RttAllEntriesS2AP(
    rtt : RmmRtt,
    rtte : RmmRttEntry) => boolean
```

B3.160 RttAllEntriesState function

Returns TRUE if all entries in the RTT at address `rtt` have state `state`.

```
readonly func RttAllEntriesState(
    rtt : RmmRtt,
    state : RmmRttEntryState) => boolean
```

See also:

- A5.6 *Realm Translation Table*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.161 RttAt function

Returns the RTT at address `rtt`.

```
readonly func RttAt(
    addr : Address) => RmmRtt
```

B3.162 RttConfigIsValid function

Returns TRUE if the RTT configuration values provided are self-consistent and are supported by the platform.

```
readonly func RttConfigIsValid(
    ipa_width : integer,
    rtt_level_start : integer,
    rtt_num_start : integer) => boolean
```

See also:

- A5.6 *Realm Translation Table*

B3.163 RttDescriptorDecode function

Decode an RTT descriptor.

```
pure func RttDescriptorDecode(
    desc : bits(64),
    encoding : RmmRttS2APEncoding) => RmmRttEntry
```

B3.164 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.

```
pure func RttDescriptorIsValidForUnprotected(
    desc : bits(64)) => boolean
```

See also:

- A5.6.12.3 *Memory attributes for RTTE_MAPPED_NS mappings*

B3.165 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.

```
readonly 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);
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        if (!PaIsDelegableCohDevMem(rtte.addr)) then
            return FALSE;
        end;

        addr = ToAddress(UInt(addr) + size);
    end;

    return TRUE;
end;
```

B3.166 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`.

```
readonly 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;
```

B3.167 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.

```
readonly 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
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            return FALSE;
        end;

        addr = ToAddress(UInt(addr) + size);
    end;

    return TRUE;
end;
```

B3.168 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`.

```
readonly 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;
```

B3.169 RttEntriesInRangeRipas function

Returns TRUE if all entries in the RTT at address `rtt` at level `level`, within IPA range [base, top), have RIPAS `ripas`.

```
readonly 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);
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        if (rtte.ripas != ripas) then
            return FALSE;
        end;

        addr = ToAddress(UInt(addr) + size);
    end;

    return TRUE;
end;
```

B3.170 RttEntryAt function

Returns the `i`th entry in the RTT `rtt`.

```
readonly func RttEntryAt(
    rtt : RmmRtt,
    i : integer) => RmmRttEntry
```

See also:

- A5.6 *Realm Translation Table*

B3.171 RttEntryIndex function

Returns the index of the entry in a level `level` RTT which is identified by `addr`.

```
readonly func RttEntryIndex(
    addr : Address,
    level : integer) => integer
```

See also:

- A5.6 *Realm Translation Table*

B3.172 RttEntryStateToRmi function

Encodes the state of an RTTE.

```
readonly func RttEntryStateToRmi(
    state : RmmRttEntryState) => RmiRttEntryState
begin
    case state of
        when RTTE_VOID         => return RMI_RTTE_VOID;
        when RTTE_DATA           => return RMI_RTTE_DATA;
        when RTTE_UNMAPPED_NS      => return RMI_RTTE_VOID;
        when RTTE_MAPPED_NS        => return RMI_RTTE_DATA;
        when RTTE_TABLE              => return RMI_RTTE_TABLE;
        when RTTE_NARCH_DEV       => return RMI_RTTE_NARCH_DEV;
        when RTTE_AUX_DESTROYED      => return RMI_RTTE_AUX_DESTROYED;
        when RTTE_ARCH_DEV => return RMI_RTTE_ARCH_DEV;
    end;
end;
```

B3.173 RttFold function

Returns the RTTE which results from folding the homogeneous RTT at address `rtt`.

```
pure func RttFold(
    rtt : RmmRtt) => RmmRttEntry
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- A5.6.6 *RTT folding*

B3.174 RttIsHomogeneous function

Returns TRUE if the RTT at address `rtt` is homogeneous.

```
pure func RttIsHomogeneous(
    rtt : RmmRtt) => boolean
```

See also:

- A5.6.6 *RTT folding*

B3.175 RttIsLive function

Returns TRUE if the RTT at address `rtt` is live.

```
pure func RttIsLive(
    rtt : RmmRtt) => boolean
```

See also:

- A5.6.8 *RTTE liveness and RTT liveness*
- A5.6.9 *RTT destruction*

B3.176 RttLevelIsStarting function

Returns TRUE if `level` is the starting level of the RTT for the Realm described by `rd`.

```
pure func RttLevelIsStarting(
    realm : RmmRealm,
    level : integer) => boolean
```

See also:

- A5.6 *Realm Translation Table*

B3.177 RttLevelIsValid function

Returns TRUE if `level` is a valid RTT level for the Realm described by `rd`.

```
pure func RttLevelIsValid(
    realm : RmmRealm,
    level : integer) => boolean
```

See also:

- A5.6 *Realm Translation Table*

B3.178 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
readonly func RttLevelSize(
    level : integer) => integer
```

See also:

- A5.6 *Realm Translation Table*

B3.179 RttMemAttrEqual function

Returns TRUE if the memory attributes of the two RTT entries match.

```
readonly 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;
```

B3.180 RttS2APEqual function

Returns TRUE if the S2AP of the two RTT entries match.

```
readonly 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;
```

B3.181 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`.

```
readonly func RttsAllProtectedEntriesRipas(
    rtt_base : Address,
    rtt_num_start : integer,
    ripas : RmmRipas) => boolean
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.182 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`.

```
readonly func RttsAllProtectedEntriesState(
    rtt_base : Address,
    rtt_num_start : integer,
    state : RmmRttEntryState) => boolean
```

B3.183 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`.

```
readonly func RttsAllUnprotectedEntriesState(
    rtt_base : Address,
    rtt_num_start : integer,
    state : RmmRttEntryState) => boolean
```

B3.184 RttsGranuleState function

Inductive function which identifies the states of the starting-level RTT Granules.

This function is used in the definition of command footprint.

```
readonly func RttsGranuleState(
    rtt_base : Address,
    rtt_num_start : integer)
```

B3.185 RttSkipEntriesIfState function

Scanning `rtt` starting from `base`, 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`.

```
readonly func RttSkipEntriesIfState(
    rtt : RmmRtt,
    level : integer,
    base : Address,
    state : RmmRttEntryState) => Address
```

B3.186 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`.

```
readonly func RttSkipEntriesUnlessRipas(
    rtt : RmmRtt,
    level : integer,
    ipa : Address,
    ripas : RmmRipas) => Address
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.187 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`.

```
readonly func RttSkipEntriesUnlessState(
    rtt : RmmRtt,
    level : integer,
    ipa : Address,
    state : RmmRttEntryState) => Address
```

B3.188 RttSkipEntriesUnlessVoidOrData function

Scan `rtt` starting from `base` and terminating at `top`.

Return IPA of the first entry whose state is neither RTTE_VOID nor RTTE_DATA.

```
readonly func RttSkipEntriesUnlessVoidOrData(
    rtt : RmmRtt,
    level : integer,
    base : Address,
    top : Address) => Address
begin
    var top_void : Address = RttSkipEntriesIfState(
                rtt, level, base, RTTE_VOID);

    var top_data : Address = RttSkipEntriesIfState(
                rtt, level, base, RTTE_DATA);

    return MinAddress(
        MinAddress(top_void, top_data),
        top);
end;
```

B3.189 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 RTTE_TABLE.
- If stop_at_destroyed is TRUE then return IPA of the first entry whose state is RTTE_TABLE or whose RIPAS is RIPAS_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`.

```
readonly func RttSkipEntriesWithRipas(
    rtt : RmmRtt,
    level : integer,
    base : Address,
    top : Address,
    stop_at_destroyed : boolean) => Address
begin
    var result : Address = RttSkipEntriesUnlessState(
                rtt, level, base, RTTE_TABLE);
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    if stop_at_destroyed then
        result = MinAddress(result,
            RttSkipEntriesUnlessRipas(
                rtt, level, base, RIPAS_DESTROYED));
    end;

    result = MinAddress(result, top);

    return AlignDownToRttLevel(result, level);
end;
```

B3.190 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`.

```
readonly func RttSkipNonLiveEntries(
    rtt : RmmRtt,
    level : integer,
    ipa : Address) => Address
begin
    var result : Address = RttSkipEntriesUnlessState(
                rtt, level, ipa, RTTE_DATA);

    result = MinAddress(result,
            RttSkipEntriesUnlessState(
                rtt, level, ipa, RTTE_MAPPED_NS));

    result = MinAddress(result,
            RttSkipEntriesUnlessState(
                rtt, level, ipa, RTTE_TABLE));

    result = MinAddress(result,
            RttSkipEntriesUnlessState(
                rtt, level, ipa, RTTE_NARCH_DEV));

    return AlignDownToRttLevel(result, level);
end;
```

See also:

- A5.6.8 *RTTE liveness and RTT liveness*

B3.191 RttsStateEqual function

Returns TRUE if the state of all of the starting-level RTT Granules is equal to `state`.

```
readonly func RttsStateEqual(
    rtt_base : Address,
    rtt_num_start : integer,
    state : RmmGranuleState) => boolean
begin
    var rmm : RmmGlobal = Rmm();

    for i = 0 to rtt_num_start - 1 do
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
            var addr = (UInt(rtt_base) +
                i * rmm.dynamic.rmi_granule_size)[(ADDRESS_WIDTH-1):0];
            if (!PaIsTracked(addr)
                    || GranuleAt(addr).state != state) then
                return FALSE;
            end;
        end;
        return TRUE;
    end;
```

B3.192 RttTreeRangeAllMemAttr function

Returns TRUE if all RTT entries for RTT tree `tree` in Realm `realm` within IPA range `[base, top)` have memory attributes equal to `memattr`.

If the argument is a bits(3) then it is encoded as for `MemAttr[2:0]` in a translation table page or block descriptor.

```
readonly func RttTreeRangeAllMemAttr(
    realm : RmmRealm,
    tree : integer,
    base : Address,
    top : Address,
    memattr : RmmRttMemAttr) => boolean
```

```
readonly func RttTreeRangeAllMemAttr(
    realm : RmmRealm,
    tree : integer,
    base : Address,
    top : Address,
    memattr : bits(3)) => boolean
```

B3.193 RttTreeRangeAllMemAttrEqual function

Returns TRUE if RTT entries in Realm `realm` within IPA range `[base, top)` have the same memory attributes in RTT trees `tree_1` and `tree_2`.

```
readonly func RttTreeRangeAllMemAttrEqual(
    realm : RmmRealm,
    tree_1 : integer,
    tree_2 : integer,
    base : Address,
    top : Address) => boolean
```

B3.194 RttTreeRangeAllOaddr function

Returns TRUE if all RTT entries for RTT tree `tree` in Realm `realm` within IPA range `[base, top)` have output address equal to `addr`.

```
readonly func RttTreeRangeAllOaddr(
    realm : RmmRealm,
    tree : integer,
    base : Address,
    top : Address,
    addr : Address) => boolean
```

B3.195 RttTreeRangeAllOaddrContig function

Returns TRUE if RTT entries for RTT tree `tree` in Realm `realm` within IPA range `[base, top)` have output

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

addresses which map linearly to PA range [obase, obase + size).

```
readonly func RttTreeRangeAllOaddrContig(
    realm : RmmRealm,
    tree : integer,
    ibase : Address,
    obase : Address,
    size : integer) => boolean
```

B3.196 RttTreeRangeAllOaddrEqual function

Returns TRUE if RTT entries in Realm `realm` within IPA range `[base, top)` have the same output addresses in RTT trees `tree_1` and `tree_2`.

```
readonly func RttTreeRangeAllOaddrEqual(
    realm : RmmRealm,
    tree_1 : integer,
    tree_2 : integer,
    base : Address,
    top : Address) => boolean
```

B3.197 RttTreeRangeAllOaddrList function

Returns TRUE if RTT entries for RTT tree `tree` in Realm `realm` within IPA range `[base, top)` have output addresses which map linearly to the first `size` bytes of the address set described by an RMI Address Range List stored at PA `alist`.

```
readonly func RttTreeRangeAllOaddrList(
    realm : RmmRealm,
    tree : integer,
    ibase : Address,
    alist : Address,
    size : integer) => boolean
```

B3.198 RttTreeRangeAllS2AP function

Returns TRUE if all RTT entries for RTT tree `tree` in Realm `realm` within IPA range `[base, top)` have S2AP equal to `s2ap`.

If the Realm uses S2AP direct encoding, this is encoded as for `AP` in a translation table page or block descriptor.

If the Realm uses S2AP indirect encoding, this is encoded as for `PIIndex` in a translation table page or block descriptor.

```
readonly func RttTreeRangeAllS2AP(
    realm : RmmRealm,
    tree : integer,
    base : Address,
    top : Address,
    s2ap : bits(4)) => boolean
```

B3.199 RttTreeRangeAllShareability function

Returns TRUE if all RTT entries for RTT tree `tree` in Realm `realm` within IPA range `[base, top)` have shareability equal to `shareability`.

```
readonly func RttTreeRangeAllShareability(
    realm : RmmRealm,
    tree : integer,
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    base : Address,
    top : Address,
    shareability : RmmRttShareability) => boolean
```

B3.200 RttTreeRangeAllShareabilityEqual function

Returns TRUE if RTT entries in Realm `realm` within IPA range `[base, top)` have the same shareability in RTT trees `tree_1` and `tree_2`.

```
readonly func RttTreeRangeAllShareabilityEqual(
    realm : RmmRealm,
    tree_1 : integer,
    tree_2 : integer,
    base : Address,
    top : Address) => boolean
```

B3.201 RttTreeRangeAllState function

Returns TRUE if all RTT entries for RTT tree `tree` in Realm `realm` within IPA range `[base, top)` have state equal to `state`.

```
readonly func RttTreeRangeAllState(
    realm : RmmRealm,
    tree : integer,
    base : Address,
    top : Address,
    state : RmmRttEntryState) => boolean
```

B3.202 RttWalk function

Returns the result of an RTT walk from the base of RTT tree `tree` owned by `rd`, to address `addr`.

The walk does not progress beyond `level`.

```
readonly func RttWalk(
    realm : RmmRealm,
    addr : Address,
    level : integer,
    tree : integer) => RmmRttWalkResult
```

See also:

- A5.6.10 *RTT walk*

B3.203 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.

```
readonly func RttWalkAnyNotAligned(
    realm : RmmRealm,
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    base : Address,
    top : Address,
    level : integer) => RmmRttWalkNotAligned
```

B3.204 TdiIdIsFree function

Returns TRUE if `tdi_id` is unused within the namespace identified by `routing_id`.

```
readonly func TdiIdIsFree(
    tdi_id : bits(64),
    routing_id : bits(64)) => boolean
```

B3.205 ToAddress function

Convert integer to Address.

```
readonly func ToAddress(value : integer) => Address
begin
    return value[(ADDRESS_WIDTH-1):0];
end;
```

B3.206 ToBits64 function

Convert integer to Bits64.

```
pure func ToBits64(value : integer) => bits(64)
begin
    return value[63:0];
end;
```

B3.207 TrackingRegionAt function

Returns tracking region whose base address is `addr`.

```
readonly func TrackingRegionAt(
    addr : Address) => RmmTrackingRegion
```

B3.208 TrackingRegionGranularity function

Returns granularity at which specified region is tracked.

```
readonly func TrackingRegionGranularity(
    region : RmmTrackingRegion) => integer
begin
    var rmm : RmmGlobal = Rmm();

    assert TrackingRegionIsTracked(region);

    case region.state of
        when TRACKING_COARSE => return rmm.dynamic.tracking_region_size;
        when TRACKING_FINE   => return rmm.dynamic.rmi_granule_size;
    end;
end;
```

B3.209 TrackingRegionIsTracked function

Returns TRUE if the tracking region is tracked.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
readonly func TrackingRegionIsTracked(
    region : RmmTrackingRegion) => boolean
begin
    return region.state IN {
        TRACKING_COARSE,
        TRACKING_FINE
    };
end;
```

B3.210 TrackingRegionSizeFromRmi function

Decodes a tracking region size.

```
readonly func TrackingRegionSizeFromRmi(
    granule_size : RmiGranuleSize,
    tracking_region_size : integer) => integer
begin
    case granule_size of
        when RMI_GRANULE_SIZE_4KB =>
            case tracking_region_size of
                when 0    => return   1 * GB; // Level 1 block size
                otherwise => return 0;
            end;

        when RMI_GRANULE_SIZE_16KB =>
            case tracking_region_size of
                when 0    => return  32 * MB; // Level 2 block size
                when 1    => return  64 * GB; // Level 1 block size
                otherwise => return 0;
            end;

        when RMI_GRANULE_SIZE_64KB =>
            case tracking_region_size of
                when 0    => return 512 * MB; // Level 2 block size
                when 1    => return   4 * TB; // Level 1 block size
                otherwise => return 0;
            end;
    end;
end;
```

B3.211 TrackingRegionSizeToRmi function

Encodes a block size.

```
readonly func TrackingRegionSizeToRmi(
    granule_size : integer,
    tracking_region_size : integer) => integer
begin
    case granule_size of
        when 4 * KB =>
            case tracking_region_size of
                when   1 * GB => return 0; // Level 1 block size
                otherwise     => unreachable;
            end;

        when 16 * KB =>
            case tracking_region_size of
                when  32 * MB => return 0; // Level 2 block size
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
                when  64 * GB => return 1; // Level 1 block size
                otherwise     => unreachable;
            end;

        when 64 * KB =>
            case tracking_region_size of
                when 512 * MB => return 0; // Level 2 block size
                when   4 * TB => return 1; // Level 1 block size
                otherwise     => unreachable;
            end;
    end;
end;
```

B3.212 VdevAddrInRange function

Returns TRUE if `addr` is within the address ranges of `vdev`.

```
readonly func VdevAddrInRange(
    addr : Address,
    vdev : RmmVdev) => boolean
begin
    for i = 0 to vdev.num_addr_range - 1 do
        var range : RmmAddrRange = vdev.addr_range[[i]];
        if AddrIsWithin(addr, range.base, range.top) then
            return TRUE;
        end;
    end;

    return FALSE;
end;
```

B3.213 VdevAt function

Returns the VDEV object located at physical address `addr`.

```
readonly func VdevAt(
    addr : Address) => RmmVdev
```

B3.214 VdevAttestInfoEqual function

Returns TRUE if the VDEV attestation info matches.

```
pure func VdevAttestInfoEqual(
    lock_nonce : integer,
    meas_nonce : integer,
    report_nonce : integer,
    attest_info : RmmVdevAttestInfo) => boolean
begin
    return (lock_nonce == attest_info.lock_nonce
        && meas_nonce == attest_info.meas_nonce
        && report_nonce == attest_info.report_nonce);
end;
```

```
pure func VdevAttestInfoEqual(
    attest_info_1 : RmmVdevAttestInfo,
    attest_info_2 : RmmVdevAttestInfo) => boolean
begin
    return (attest_info_1.lock_nonce == attest_info_2.lock_nonce
        && attest_info_1.meas_nonce == attest_info_2.meas_nonce
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
        && attest_info_1.report_nonce == attest_info_2.report_nonce);
end;
```

B3.215 VdevFindMapped function

Returns TRUE if any Granule within the address ranges of `vdev` is mapped.

```
readonly func VdevFindMapped(
    vdev : RmmVdev) => RmmVdevAddrResult
begin
    var rmm : RmmGlobal = Rmm();

    for i = 0 to vdev.num_addr_range - 1 do
        var range : RmmAddrRange = vdev.addr_range[[i]];
        var addr : Address = range.base;
        while UInt(addr) < UInt(range.top) do
            if GranuleAt(addr).state == GRAN_DEV then
                return RmmVdevAddrResult { valid = RMM_TRUE, addr = addr };
            end;
            addr = ToAddress(UInt(addr) + rmm.dynamic.rmi_granule_size);
        end;
    end;

    return RmmVdevAddrResult {
        valid = RMM_FALSE, addr = ARBITRARY : Address };
end;
```

B3.216 VdevFromVdevId function

Returns the VDEV identified by `vdev_id` and assigned to `realm`.

```
readonly func VdevFromVdevId(
    realm : RmmRealm,
    vdev_id : bits(64)) => RmmVdev
```

B3.217 VdevGenerateNonce function

Generate a VDEV nonce.

```
readonly impdef func VdevGenerateNonce(
    vdev : RmmVdev) => integer
```

B3.218 VdevIdIsFree function

Returns TRUE if `vdev_id` does not identify any device which is assigned to `realm`.

```
readonly func VdevIdIsFree(
    realm : RmmRealm,
    vdev_id : bits(64)) => boolean
```

B3.219 VdevSid function

Returns physical SID for the specified VDEV.

```
readonly func VdevSid(
    vdev : RmmVdev) => bits(64)
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.220 VdevStateToRsi function

Get VDEV state

```
readonly func VdevStateToRsi(
    vdev_state : RmmVdevState) => RsiVdevState
begin
    case vdev_state of
        when VDEV_NEW         => return RSI_VDEV_UNLOCKED;
        when VDEV_UNLOCKED    => return RSI_VDEV_UNLOCKED;
        when VDEV_LOCKED      => return RSI_VDEV_LOCKED;
        when VDEV_STARTED     => return RSI_VDEV_STARTED;
        when VDEV_ERROR       => return RSI_VDEV_ERROR;
        when VDEV_KEY_REFRESH => return RSI_VDEV_STARTED;
        when VDEV_KEY_PURGE   => return RSI_VDEV_STARTED;
    end;
end;
```

B3.221 VersionEqual function

Returns TRUE if command result matches the stated value.

```
pure func VersionEqual(
    ver1 : PsciInterfaceVersion,
    ver2 : PsciInterfaceVersion) => boolean
```

```
pure func VersionEqual(
    ver1 : RmiInterfaceVersion,
    ver2 : RmiInterfaceVersion) => boolean
```

```
pure func VersionEqual(
    ver1 : RsiInterfaceVersion,
    ver2 : RsiInterfaceVersion) => boolean
```

See also:

- Chapter B2 *Interface versioning*

B3.222 VmidsAvailable function

Returns TRUE if there are `count` unused VMIDs.

```
readonly func VmidsAvailable(
    count : integer) => boolean
```

See also:

- A5.6.2 *RTT structure and configuration*

B3.223 VsidIsFree function

Returns TRUE if `vsid` is unused within the VSMMU `vsmmu`.

```
readonly func VsidIsFree(
    vsmmu : RmmVsmmu,
    vsid : bits(64)) => boolean
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B3.224 VsmmuAt function

Returns the VSMMU object located at physical address `addr`.

```
readonly func VsmmuAt(
    addr : Address) => RmmVsmmu
```

See also:

- A9.8 *Virtual SMMU*

B3.225 VsmmuFeaturesAt function

Returns the RmiVsmmuFeatures object located at physical address `addr`.

```
readonly func VsmmuFeaturesAt(
    addr : Address) => RmiVsmmuFeatures
```

B3.226 VsmmuIsLive function

Returns TRUE if the VSMMU located at physical address `addr` is live.

```
readonly func VsmmuIsLive(
    addr : Address) => boolean
```

See also:

- A9.8.3 *VSMMU liveness*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter B4
Realm Management Interface

DRAFT

This chapter defines the interface used by the Host to manage Realms.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.1 RMI version

R_{NCFDX} This specification defines version 2.0 of the Realm Management Interface.

See also:

- Chapter B2 *Interface versioning*
- B4.5.82 *RMI_VERSION command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.2 RMI command return codes

IYCHQV The *status* field of an RmiResult value indicates the outcome of the command, which is one of the following:

Outcome	System state	Description	Return status
Success	Complete transition	The command completed successfully, leaving no intermediate state. Either: • A single-object command has transitioned the target object to the destination state. • A range operation has transitioned a subset of the target range to the destination state. For example, given a request for a transition from “A” to “B”, the command has transformed “A A A A” to “B B A A”. The extent of the subset which has transitioned to the destination state may be indicated by the command output values. The command did not leave any object in an intermediate state.	RMI_SUCCESS
	Incomplete transition	The command initiated a state transition but did not complete, leaving an object in an intermediate state. Either: • A non-range operation has transitioned the target object to an intermediate state “X”. For example, given a request for a transition from “A” to “B”, the command has transformed “A” to “X”. • A range operation has transitioned a subset of the target range to the destination state, and has transitioned a further subset to an intermediate state. For example, given a request for a transition from “A” to “B”, the command has transformed “A A A A” to “B B X A”. The extent of the subset which has transitioned to the destination state may be. indicated by the command output values. The extent of the subset which has transitioned to the intermediate state is UNKNOWN.	RMI_INCOMPLETE
Failure	Invalid input or invalid state	The command failed to make progress due to invalid input values, or inconsistency between the input values and the system state. The command did not result in any changes of system state.	RMI_ERROR_*
	Blocked	The command failed to make progress due to a target object being in an intermediate state. The intermediate state will persist until an action is taken by the Host. The command did not result in any changes of system state.	RMI_BLOCKED

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Outcome	System state	Description	Return status
	Busy	The command failed to make progress, for an IMPLEMENTATION DEFINED reason. The reason for the lack of progress may be temporary, and may be resolved without requiring any action to be taken by the Host. The command did not result in any changes of system state.	RMI_BUSY

I_{WNPWC} The encoding of the remaining bits of an RmiResult value depend on the value of the *status* field.

I_{MBVPG} For some RmiResult status values, a *level* field provides additional information about the reason for a command failure. These are described in the following table.

Status	Description	Meaning of level
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.
RMI_ERROR_PSMMU_ST	An PSMMU Stream Table walk terminated before reaching the target level, or reached an entry with an unexpected value	PSMMU Stream Table level at which the walk terminated.
RMI_ERROR_DPT	A DPT walk terminated before reaching the target level, or reached an entry with an unexpected value	DPT level at which the walk terminated.

DRAFT

I_{WKFTV} If the RmiResult staus value is RMI_INCOMPLETE then the *handle* field contains an IMPLEMENTATION DEFINED value which identifies the incomplete RMI operation.

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:

- B4.3.2 *Stateful RMI operations*
- B4.6.73 *RmiResult type*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.3 RMI operations

B4.3.1 RMI operations overview

D_{ZLDRW} An *RMI operation* is an operation which is performed in response to execution of an RMI command.

D_{DHQDH} An *initiating RMI command* is a request for the RMM to initiate an RMI operation.

D_{KYLMC} The inputs of an initiating RMI command define its *target object scope*, which can be one of the following:

- A *range RMI command* has inputs which include an address range (IPA or PA). The address range identifies a set of objects on which the command will operate.
- A *non-range RMI command* has inputs among which any addresses (IPA or PA) are single values. Each address identifies a single object on which the command will operate.

D_{BLCSV} The *statefulness* property of an RMI operation is one of the following:

- A *stateful RMI operation* is an operation for which the RMM may maintain state in an internal data structure. The internal state is identified by an RMM-allocated handle which is returned to the caller. Completion of the operation requires multiple RMI calls.
- A *stateless RMI operation* is an operation for which the RMM does not maintain internal state. Its actions are limited to modifying state in objects which have been allocated using RMI commands (for example REC or RTT). The operation is completed during execution of the initiating RMI command.

D_{RHMXQ} The *memory transfer* property of an RMI operation is one of the following:

- A *memory-transferring RMI operation* is one which may cause the RMM to request the caller to either donate or reclaim memory, in order for the operation to progress.
- A *non-memory-transferring RMI operation* is one which does not cause the RMM to request the caller to donate or reclaim memory.

R_{FXPVW} A memory-transferring operation is stateful.

I_{GTRHX} The definition of each initiating RMI command includes the following:

- Whether the initiated RMI operation can be memory-transferring.
- Whether the initiated RMI peration can be stateful.

See also:

- B4.3.2 *Stateful RMI operations*
- B4.3.3 *RMI operations which can lead to an intermediate state*
- B4.3.5 *Range RMI operations*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.3.2 Stateful RMI operations

U_{FBVXR} When an RMI operation cannot be completed within an IMPLEMENTATION DEFINED time limit, the implementation may use a *Stateful RMI Operation (SRO)* to save the state of the operation before returning to the Host.

U_{PMRCX} The time budget for an RMI operation may depend upon the state of the system. For example, an RMI operation may be:

- permitted to continue execution for an unbounded amount of time while no interrupts are pending, and
- guaranteed to yield within an IMPLEMENTATION DEFINED time limit from the point at which an interrupt becomes pending.

U_{LFSBZ} Usage of SROs allows an implementation to ensure forward progress while limiting added interrupt latency to an IMPLEMENTATION DEFINED upper bound.

U_{TTPZC} Usage of SROs allows an implementation to require that memory is transferred between Host and RMM before an RMI operation can continue.

D_{LVXCB} Properties of an SRO are stored in an RMM-internal data structure called an *SRO context*.

D_{ZXLQF} An *SRO handle* is an IMPLEMENTATION DEFINED value which is generated by the RMM and which identifies an SRO context.

R_{HQSCC} When an RMI command has initiated an SRO, all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::cancel indicates whether the SRO can be cancelled.
- RmiResult::mem indicates whether the SRO is memory-transferring.
- X1 holds the SRO handle.
- If RmiResult::mem is RMI_OP_MEM_REQ_DONATE then X2 holds an RmiOpMemDonateReq value.

I_{YRKZY} An SRO context stores all Host-provided information which is necessary to complete the operation.

U_{RGDMZ} Allocation of an SRO context may fail for the following reasons:

- All SRO contexts have been allocated, and none of the corresponding operations are currently running - in other words, all of the RMI commands which operate on that SRO have completed and returned to the Host. In this case, the allocation failure is reported by returning RMI_BLOCKED.
- All SRO contexts have been allocated, and at least one of the corresponding operations is currently running. In this case, the allocation failure is reported by returning RMI_BUSY.

I_{JVYCH} The following table lists the RMI commands which can initiate an SRO, and for each one states whether the SRO can be memory-transferring.

Command	Stateful	Memory-transferring
RMI_DPT_L0_CREATE	Y	Y
RMI_DPT_L0_DESTROY	Y	Y
RMI_DPT_L1_CREATE	Y	Y
RMI_DPT_L1_DESTROY	Y	Y
RMI_FEATURES		
RMI_GPT_L1_CREATE	Y	Y
RMI_GPT_L1_DESTROY	Y	Y
RMI_GRANULE_RANGE_DELEGATE	Y	
RMI_GRANULE_RANGE_UNDELEGATE	Y	

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Command	Stateful	Memory-transferring
RMI_GRANULE_TRACKING_GET		
RMI_GRANULE_TRACKING_SET	Y	Y
RMI_OP_CANCEL		
RMI_OP_CONTINUE		
RMI_OP_MEM_DONATE		
RMI_OP_MEM_RECLAIM		
RMI_PDEV_ABORT		
RMI_PDEV_COMMUNICATE		
RMI_PDEV_CREATE	Y	Y
RMI_PDEV_DESTROY	Y	Y
RMI_PDEV_GET_STATE		
RMI_PDEV_SET_PUBKEY		
RMI_PDEV_STOP		
RMI_PDEV_STREAM_COMPLETE		
RMI_PDEV_STREAM_CONNECT		
RMI_PDEV_STREAM_DISCONNECT		
RMI_PDEV_STREAM_KEY_PURGE		
RMI_PDEV_STREAM_KEY_REFRESH		
RMI_PSCI_COMPLETE		
RMI_PSMMU_ACTIVATE	Y	Y
RMI_PSMMU_DEACTIVATE	Y	Y
RMI_PSMMU_EVENT_CONSUME		
RMI_PSMMU_INFO		
RMI_PSMMU_IRQ_NOTIFY		
RMI_PSMMU_ST_L2_CREATE	Y	Y
RMI_PSMMU_ST_L2_DESTROY	Y	Y
RMI_REALM_ACTIVATE		
RMI_REALM_CREATE	Y	Y
RMI_REALM_DESTROY	Y	Y
RMI_REALM_TERMINATE	Y	
RMI_REC_CREATE	Y	Y
RMI_REC_DESTROY	Y	Y
RMI_REC_ENTER		
RMI_RMM_ACTIVATE	Y	Y
RMI_RMM_CONFIG_GET		

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Command	Stateful	Memory-transferring
RMI_RMM_CONFIG_SET		
RMI_RMM_STATE_GET		
RMI_RTT_ARCH_DEV_MAP		
RMI_RTT_ARCH_DEV_UNMAP		
RMI_RTT_AUX_CREATE	Y	
RMI_RTT_AUX_DESTROY	Y	
RMI_RTT_AUX_FOLD	Y	
RMI_RTT_AUX_PROT_MAP	Y	
RMI_RTT_AUX_PROT_UNMAP	Y	
RMI_RTT_AUX_UNPROT_MAP	Y	
RMI_RTT_AUX_UNPROT_UNMAP	Y	
RMI_RTT_CREATE	Y	
RMI_RTT_DATA_MAP	Y	
RMI_RTT_DATA_MAP_INIT	Y	
RMI_RTT_DATA_UNMAP	Y	
RMI_RTT_DESTROY	Y	
RMI_RTT_DEV_MAP	Y	
RMI_RTT_DEV_UNMAP	Y	
RMI_RTT_DEV_VALIDATE	Y	
RMI_RTT_FOLD	Y	
RMI_RTT_INIT_RIPAS	Y	
RMI_RTT_READ_ENTRY		
RMI_RTT_SET_RIPAS	Y	
RMI_RTT_SET_S2AP	Y	
RMI_RTT_UNPROT_MAP	Y	
RMI_RTT_UNPROT_UNMAP	Y	
RMI_VDEV_ABORT		
RMI_VDEV_COMMUNICATE		
RMI_VDEV_COMPLETE		
RMI_VDEV_CREATE	Y	Y
RMI_VDEV_DESTROY	Y	Y
RMI_VDEV_GET_INTERFACE_REPORT		
RMI_VDEV_GET_MEASUREMENTS		
RMI_VDEV_GET_STATE		
RMI_VDEV_LOCK		

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Command	Stateful	Memory-transferring
RMI_VDEV_START		
RMI_VDEV_UNLOCK		
RMI_VERSION		
RMI_VSMMU_CMD_COMPLETE		
RMI_VSMMU_CMD_GET		
RMI_VSMMU_CREATE	Y	Y
RMI_VSMMU_DESTROY	Y	Y
RMI_VSMMU_EVENT_HANDLE		
RMI_VSMMU_FEATURES		

See also:

- A2.3.4 *Granule tracking region*
- A9.7.1.2 *PSMMU lifecycle*
- B3.47 *OperationCanCancel function*
- B3.48 *OperationIncomplete function*
- B4.3.4 *Object creation and destruction*
- B4.6.73 *RmiResult type*

DRAFT

B4.3.2.1 Continuing an SRO

R_{SFNPS} If an SRO is ready to continue then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::cancel indicates whether the SRO can be cancelled.
- The SRO handle is valid.

R_{RZLLJ} If RmiResult::status is RMI_INCOMPLETE then RmiResult::mem indicates which command is required to continue the SRO:

- RMI_OP_MEM_REQ_NONE indicates that RMI_OP_CONTINUE is required.
- RMI_OP_MEM_REQ_DONATE indicates that RMI_OP_MEM_DONATE is required.
- RMI_OP_MEM_REQ_RECLAIM indicates that RMI_OP_MEM_RECLAIM is required.

I_{DKTKY} The input values of RMI_OP_CONTINUE include an SRO handle.

R_{KLDWG} On execution of RMI_OP_CONTINUE, if the SRO has completed successfully then all of the following are true:

- RmiResult::status is RMI_SUCCESS.
- Output values X1..Xn are populated as specified by the initiating command.
- The SRO handle is no longer valid.

R_{YFGZH} On execution of RMI_OP_CONTINUE, if the SRO has completed with an error then all of the following are true:

- RmiResult::status is a value other than RMI_BUSY, RMI_INCOMPLETE and RMI_SUCCESS.
- Output values X1..Xn are populated as specified by the initiating command.
- The SRO handle is no longer valid.

See also:

- B4.3.2.2 *Donating memory to an SRO*
- B4.3.2.3 *Reclaiming memory from an SRO*
- B4.5.13 *RMI_OP_CONTINUE command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.3.2.2 Donating memory to an SRO

R_{JDQHX} If an SRO requires memory to be donated before it can continue then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::mem is RMI_OP_MEM_REQ_DONATE.
- X2 contains an RmiOpMemDonateReq value which describes memory which must be donated to continue the SRO, encoded as shown in the following table.

Name	Bits	Description	Value
size	1:0	Size of each block	RmiAddrBlockSize
count	15:2	Number of blocks required	UInt14
contig	16	Whether blocks are required to be contiguous	RmiOpMemContig
state	17	Required initial state of memory	RmiOpMemState
	63:18	Reserved	SBZ

I_{FJCVF} Donation of memory to an SRO is performed by execution of RMI_OP_MEM_DONATE.

I_{GZNYB} The input values of RMI_OP_MEM_DONATE include an SRO handle.

I_{LJBYD} The input values of RMI_OP_MEM_DONATE include a pointer to an *RMI Address Range List*, which is a data structure that contains a list of *RMI Address Range Descriptors*, each of which describes a region of contiguous address space.

I_{QMVVQ} The output values of RMI_OP_MEM_DONATE include the number of Granules which have been successfully donated to the SRO.

R_{MDVYC} On execution of RMI_OP_MEM_DONATE, the `donated_count` output value is valid regardless of the RmiResult::status value.

R_{YQVYM} On execution of RMI_OP_MEM_DONATE, if the amount of memory described by the input RMI Address Range List is greater than the amount required then RmiResult::status is RMI_ERROR_INPUT.

R_{TFSKF} On execution of RMI_OP_MEM_DONATE, if the requirement was for contiguous memory and the memory described by the input RMI Address Range List is not contiguous then RmiResult::status is RMI_ERROR_INPUT.

R_{NLGPW} On execution of RMI_OP_MEM_DONATE, if the state of memory described by the input RMI Address Range List does not match the required state then RmiResult::status is RMI_ERROR_INPUT.

R_{GHRLW} On execution of RMI_OP_MEM_DONATE, if the input address list includes a pointer to a Granule whose state does not match the required state then RmiResult::status is RMI_ERROR_INPUT.

S_{CHVJV} If RMI_OP_MEM_DONATE returns an RmiResult::status value which is RMI_ERROR_* then the Host is expected to call RMI_OP_CONTINUE.

R_{WNDTG} On execution of RMI_OP_MEM_DONATE, if the donation was successful but additional memory must be donated to continue the SRO then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::mem is RMI_OP_MEM_REQ_DONATE.

R_{FTHBC} On execution of RMI_OP_MEM_DONATE, if the donation was successful and no additional memory must be donated to continue the SRO then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::mem is RMI_OP_MEM_REQ_NONE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{VWMYP} When all memory required by an SRO has been donated, execution of RMI_OP_CONTINUE is required to continue the SRO.

See also:

- B4.3.2.1 *Continuing an SRO*
- B4.4 *RMI address range*
- B4.5.14 *RMI_OP_MEM_DONATE command*
- B4.6.39 *RmiOpMemDonateReq type*

B4.3.2.3 Reclaiming memory from an SRO

R_{JWHXM} If an SRO requires memory to be reclaimed before it can continue or complete then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::mem is RMI_OP_MEM_REQ_RECLAIM.

I_{HLDTQ} Reclamation of memory from an SRO is performed by execution of RMI_OP_MEM_RECLAIM.

I_{YSVPS} The input values of RMI_OP_MEM_RECLAIM include an SRO handle.

I_{FMHPZ} The input values of RMI_OP_MEM_RECLAIM include a pointer to an empty RMI Address Range List.

R_{XQLVQ} On execution of RMI_OP_MEM_RECLAIM, if the reclamation is successful and additional memory must be reclaimed to continue or complete the SRO then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::mem is RMI_OP_MEM_REQ_RECLAIM.
- The RMI Address Range List is populated with a description of the memory that has been reclaimed.
- An output value indicates the number of entries in the list which are valid.
- Each entry in the list includes a flag which indicates whether the reclaimed memory is delegated or undelegated.
- The SRO handle is valid.

R_{HDKKY} On execution of RMI_OP_MEM_RECLAIM, if the reclamation is successful and no additional memory must be reclaimed to continue or complete the SRO then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::mem is RMI_OP_MEM_REQ_NONE.
- The RMI Address Range List is populated with a description of the memory that has been reclaimed.
- An output value indicates the number of entries in the list which are valid.
- Each entry in the list includes a flag which indicates whether the reclaimed memory is delegated or undelegated.
- The SRO handle is valid.

I_{ZQGGL} When all available memory has been reclaimed from an SRO, execution of RMI_OP_CONTINUE is required to continue or complete the SRO.

See also:

- B4.3.2.1 *Continuing an SRO*
- B4.4 *RMI address range*
- B4.5.15 *RMI_OP_MEM_RECLAIM command*

B4.3.2.4 Cancelling an SRO

R_{NCBJG} If an SRO can be cancelled then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::cancel is RMI_OP_CAN_CANCEL.

I_{MDBWZ} Cancelling an SRO is performed by execution of RMI_OP_CANCEL.

I_{WQHCB} The input values of RMI_OP_CANCEL include an SRO handle.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{RVLNV} On execution of RMI_OP_CANCEL, if the SRO has been cancelled but memory must be reclaimed before the SRO context can be freed then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::mem is RMI_OP_MEM_REQ_RECLAIM.
- The SRO handle is valid.

R_{GGSTJ} On execution of RMI_OP_CANCEL, if the SRO has been cancelled and no memory must reclaimed before the SRO context can be freed then all of the following are true:

- RmiResult::status is RMI_INCOMPLETE.
- RmiResult::mem is RMI_OP_MEM_REQ_NONE.
- The SRO handle is valid.

R_{LFTWL} Following successful execution of RMI_OP_CANCEL, the SRO does not request donation of memory.

I_{WLRYM} When an SRO has been cancelled and all available memory has been reclaimed, execution of RMI_OP_CONTINUE is required to make the SRO handle invalid.

I_{TWLYL} If an SRO cannot be cancelled then execution of RMI_OP_CONTINUE is required to make the SRO handle invalid.

See also:

- B4.3.2.3 *Reclaiming memory from an SRO*
- B4.5.12 *RMI_OP_CANCEL command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.3.2.5 Programming model for an SRO

I_JHFJM The following pseudocode shows the programming model for using an SRO, taking RMI_REALM_CREATE as the example initiating command.

```
/*
 * Create a Realm
 *
 * params: PA of an GRAN_UNDELEGATED Granule which contains Realm parameters
 */
int create_realm(uint64_t params)
{
    struct RmiResult result;
    struct RmiOpMemDonateReq donate_req;
    struct RmiAddrRangeDesc list[LIST_SIZE];
    uint64_t rd = alloc_and_delegate(1, RMI_BLOCK_L3);

    do {
        // Initiate the SRO.
        result = RMI_REALM_CREATE(rd, params, &donate_req);
    } while (result.status == RMI_BUSY);

    while (true) {
        if (result.status == RMI_INCOMPLETE) {
            uint64_t handle = X1;

            if (result.cancel == RMI_OP_CAN_CANCEL
                && host_decided_to_cancel()) {
                // Host decided to cancel the SRO.
                result = RMI_OP_CANCEL(handle);
            } else {
                // Host did not cancel the SRO
                if (result.mem) {
                    if (result.mem == RMI_OP_MEM_REQ_DONATE) {
                        // Call helper function which allocates memory, delegates
                        // it (if required) and then calls RMI_OP_MEM_DONATE.
                        result = donate_mem(
                            handle,
                            donate_req.contig, donate_req.state,
                            donate_req.count, donate_req.size,
                            &list[0], LIST_SIZE,
                            &donate_req
                        );
                    } else if (result.mem == RMI_OP_MEM_REQ_RECLAIM) {
                        // Call helper function calls RMI_OP_MEM_RECLAIM, then
                        // undelegates (if required) and frees the memory.
                        result = reclaim_mem(
                            handle,
                            &list[0], LIST_SIZE,
                            &donate_req
                        );
                    }

                    if (result.status != RMI_BUSY
                        && result.status != RMI_INCOMPLETE) {
                        // Memory transfer failed; allow RMM to record an error in
                        // the SRO context, return memory via RMI_OP_MEM_RECLAIM,
                        // and then report the failure of the operation to the
                        // Host via the final call to RMI_OP_CONTINUE.
                        result = RMI_OP_CONTINUE(handle, &donate_req);
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
                        }
                    } else {
                        result = RMI_OP_CONTINUE(handle, &donate_req);
                    } /* result.mem */
                }
            } /* result.status == RMI_INCOMPLETE */

            else if (result.status == RMI_BUSY) {
                result = RMI_OP_CONTINUE(handle, &donate_req);
            }

            else {
                break;
            }
        } /* true */

        return result.status;
    }
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.3.3 RMI operations which can lead to an intermediate state

I_{MGJNT} A stateful RMI operation may leave one or more objects in an *intermediate state*.

I_{WBXFG} In order for an object to exit an intermediate state, the incomplete RMI operation must be completed.

R_{BPKWX} While an object is in an intermediate state, an attempt by any RMI command to access it, except for continuation of the incomplete operation, results the command failing with a return value of RMI_BLOCKED.

See also:

- B4.3.2 *Stateful RMI operations*

B4.3.4 Object creation and destruction

I_{PFGPW} Creation and destruction of the following objects are stateful RMI operations:

- Realm Descriptor (RD)
- Realm Execution Context (REC)
- Physical device (PDEV)
- Virtual device (VDEV)
- Virtual SMMU (VSMMU)

D_{RYMMW} An RMM object is identified by the PA of its *head Granule*.

R_{QHYPW} On creation of an RMM object, if the implementation requires memory to be donated in addition to the head Granule, then the “donating memory to a Stateful RMI Operation (SRO)” flow is followed.

I_{JZTNP} The following sequence illustrates how memory donation is used during RMM object creation.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

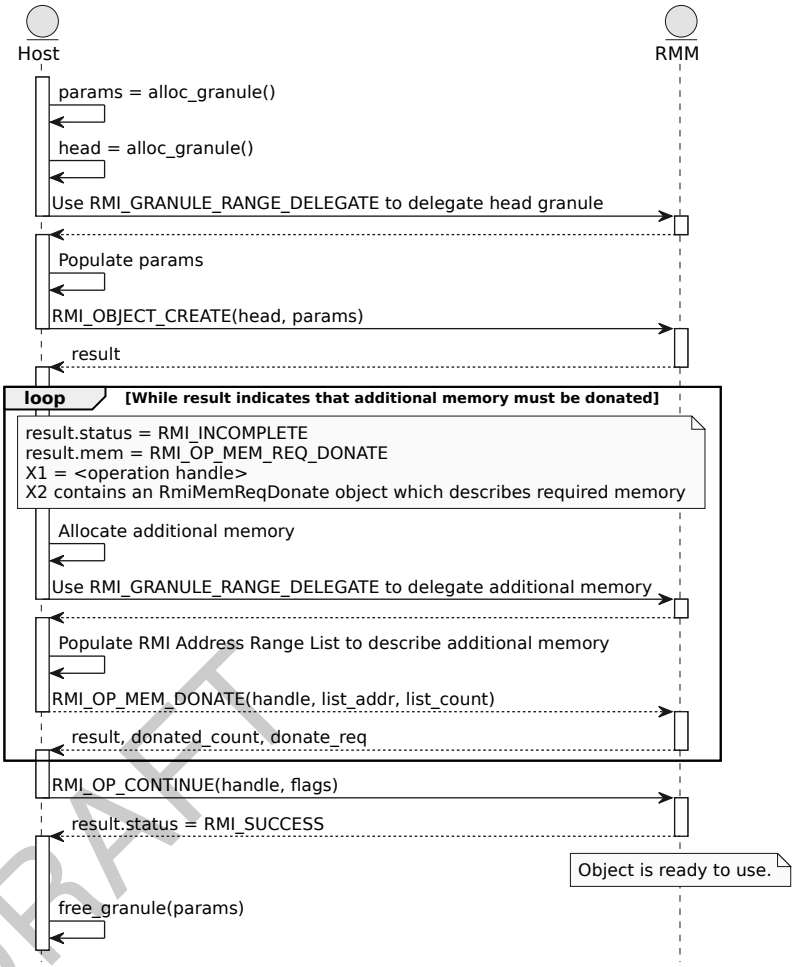


Figure B4.1: RMM object creation flow

DRAFT

R_{ZWPLQ} On destruction of an RMM object, if memory was donated during object construction then it is reclaimed via the "reclaiming memory from a Stateful RMI Operation (SRO)" flow.

I_{HZFQR} The following sequence illustrates how memory reclamation is used during RMM object destruction.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

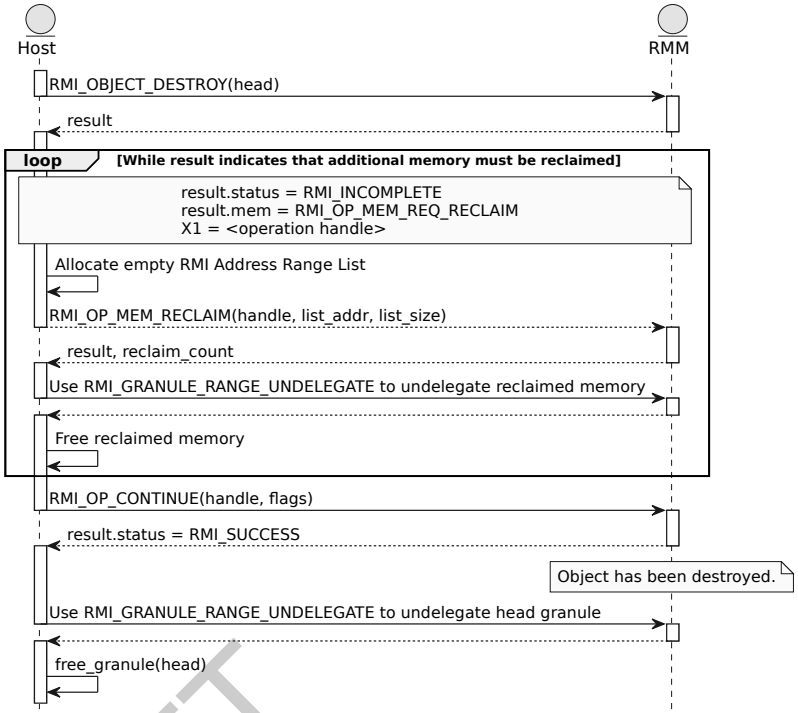


Figure B4.2: RMM object destruction flow

DRAFT

See also:

- A2.2 *Realm*
- A2.4 *Realm Execution Context*
- A9.2 *Physical device object*
- A9.4 *Virtual device object*
- A9.8 *Virtual SMMU*
- B4.3.2.2 *Donating memory to an SRO*
- B4.3.2.3 *Reclaiming memory from an SRO*

B4.3.5 Range RMI operations

I_{HNWJC} A *Range RMI operation* has the following characteristics:

- The operation modifies the state of objects which are contiguous within an address space.
- The caller initiates the operation by calling an RMI command, with the target address range identified by input values `base` and `top`.
- If the result of this command is RMI_SUCCESS then:
  - An `out_top` output value indicates the top of the target address range for which the requested state change has been completed.
  - For addresses above `out_top`, execution of the command has resulted in no state changes.
  - In order to continue the operation, the caller is expected to invoke the same command again, using `out_top` as the new `base` value. This process is repeated until `out_top == top`.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{KDSML} The following pseudocode illustrate the programming model for a Range RMI operation.

```
int rmi_range_op(
    uint64_t base, // input value 1
    uint64_t top,  // input value 2
    uint64_t in_value_3,
    ...
    uint64_t *out_top, // output value 1
    uint64_t *out_value_2,
    ...)
{
    int result;

    do {
        result = RMI_DO_OPERATION(
            base, top, in_value_3, ...,
            out_top, out_value_2, ...);
        base = *out_top;

        // If result == RMI_SUCCESS then the requested state change has been
        // applied to the range [base, *out_top).
    } while (
        result == RMI_BUSY
        || (result == RMI_SUCCESS && *out_top != top));

    // If result == RMI_SUCCESS && *out_top == top then the requested state change
    // has been applied to the entire range [base, top).

    return result;
}
```

I_{GCSXX} When a Range RMI operation returns `out_top`, the requested state transition has been applied to all objects in the range `[base, out_top)`.

I_{TRPFC} When a Range RMI operation returns `out_top`, the state of objects in the range `[out_top, top)` is unchanged by the operation.

I_{KDBMP} When a Range RMI operation returns RMI_INCOMPLETE, the Host passes a flag to RMI_OP_CONTINUE which determines how the RMM should behave once it has transitioned all objects in the target set from the intermediate state to the destination state:

- KEEP_GOING: continue processing the target set.
- STOP: return to the Host.

See also:

- B4.3.2 *Stateful RMI operations*
- B4.5.1 *RMI_DPT_L0_CREATE command*
- B4.5.2 *RMI_DPT_L0_DESTROY command*
- B4.5.3 *RMI_DPT_L1_CREATE command*
- B4.5.4 *RMI_DPT_L1_DESTROY command*
- B4.5.6 *RMI_GPT_L1_CREATE command*
- B4.5.7 *RMI_GPT_L1_DESTROY command*
- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*
- B4.5.9 *RMI_GRANULE_RANGE_UNDELEGATE command*
- B4.5.13 *RMI_OP_CONTINUE command*
- B4.5.47 *RMI_RTT_ARCH_DEV_MAP command*
- B4.5.48 *RMI_RTT_ARCH_DEV_UNMAP command*
- B4.5.52 *RMI_RTT_AUX_PROT_MAP command*
- B4.5.53 *RMI_RTT_AUX_PROT_UNMAP command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.54 *RMI_RTT_AUX_UNPROT_MAP command*
- B4.5.55 *RMI_RTT_AUX_UNPROT_UNMAP command*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.61 *RMI_RTT_DEV_MAP command*
- B4.5.62 *RMI_RTT_DEV_UNMAP command*
- B4.5.63 *RMI_RTT_DEV_VALIDATE command*
- B4.5.65 *RMI_RTT_INIT_RIPAS command*
- B4.5.67 *RMI_RTT_SET_RIPAS command*
- B4.5.68 *RMI_RTT_SET_S2AP command*
- B4.5.69 *RMI_RTT_UNPROT_MAP command*
- B4.5.70 *RMI_RTT_UNPROT_UNMAP command*

B4.4 RMI address range

This section describes data types which are used to describe contiguous ranges of address space.

See also:

- A5.3.3 *Range-based memory operations*
- B4.3.2.2 *Donating memory to an SRO*
- B4.3.2.3 *Reclaiming memory from an SRO*

DRAFT

B4.4.1 RMI Address Range Descriptor

D_{GQBKX} An *RMI Address Range Descriptor* is a data type which is used to describe a single contiguous range of address space (IPA or PA).

I_{VPRZT} An RMI Address Range Descriptor may be passed as an RMI command input or output value, either directly in a register or in Non-secure memory.

I_{FCMSK} The encoding of an RMI Address Range Descriptor depends on the RMI Granule size, as shown in the following table.

Name	Condition	Fieldset
granule_4kb	`RmmGlobal().static.rmm_granule_size == RMI_GRANULE_SIZE_4KB`	RmiAddrRangeDesc4KB
granule_16kb	`RmmGlobal().static.rmm_granule_size == RMI_GRANULE_SIZE_16KB`	RmiAddrRangeDesc16KB
granule_64kb	`RmmGlobal().static.rmm_granule_size == RMI_GRANULE_SIZE_64KB`	RmiAddrRangeDesc64KB

See also:

- A2.3.1 *Granule size*
- B4.6.3 *RmiAddrRangeDesc type*

B4.4.2 RMI Address Range List

D_{WMHWQ} An *RMI Address Range List* is a data structure which contains a list of *RMI Address Range Descriptors*. An RMI Address Range List describes a set of contiguous ranges of address space (IPA or PA).

I_{WMFTF} An RMI Address Range List may be passed as an RMI command input or output value, in Non-secure memory.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

R_{DCNQB} The base address of an RMI Address Range List is aligned to the size of an RMI Address Range Descriptor.

I_{PYWKX} An RMI Address Range List does not describe the number of RMI Address Range Descriptors which it contains. This count is provided alongside the list base address.

I_{WWTLB} An RMI Address Range List may cross one or more Granule boundaries.

See also:

- B4.4.1 *RMI Address Range Descriptor*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5 RMI commands

The following table summarizes the FIDs of commands in the RMI interface.

FID	Command
0xC4000150	RMI_VERSION
0xC4000153	RMI_RTT_DATA_MAP_INIT
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
0xC4000161	RMI_RTT_READ_ENTRY
0xC4000163	RMI_RTT_DEV_VALIDATE
0xC4000164	RMI_PSCI_COMPLETE
0xC4000165	RMI_FEATURES
0xC4000166	RMI_RTT_FOLD
0xC4000168	RMI_RTT_INIT_RIPAS
0xC4000169	RMI_RTT_SET_RIPAS
0xC400016A	RMI_VSMMU_CREATE
0xC400016B	RMI_VSMMU_DESTROY
0xC400016E	RMI_RMM_CONFIG_SET
0xC400016F	RMI_PSMMU_IRQ_NOTIFY
0xC4000174	RMI_PDEV_ABORT
0xC4000175	RMI_PDEV_COMMUNICATE
0xC4000176	RMI_PDEV_CREATE
0xC4000177	RMI_PDEV_DESTROY
0xC4000178	RMI_PDEV_GET_STATE
0xC400017A	RMI_PDEV_STREAM_KEY_REFRESH
0xC400017B	RMI_PDEV_SET_PUBKEY
0xC400017C	RMI_PDEV_STOP
0xC400017D	RMI_RTT_AUX_CREATE
0xC400017E	RMI_RTT_AUX_DESTROY
0xC400017F	RMI_RTT_AUX_FOLD

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

FID	Command
0xC4000185	RMI_VDEV_ABORT
0xC4000186	RMI_VDEV_COMMUNICATE
0xC4000187	RMI_VDEV_CREATE
0xC4000188	RMI_VDEV_DESTROY
0xC4000189	RMI_VDEV_GET_STATE
0xC400018A	RMI_VDEV_UNLOCK
0xC400018B	RMI_RTT_SET_S2AP
0xC400018E	RMI_VDEV_COMPLETE
0xC40001D0	RMI_VDEV_GET_INTERFACE_REPORT
0xC40001D1	RMI_VDEV_GET_MEASUREMENTS
0xC40001D2	RMI_VDEV_LOCK
0xC40001D3	RMI_VDEV_START
0xC40001D6	RMI_VSMMU_EVENT_HANDLE
0xC40001D7	RMI_PSMMU_ACTIVATE
0xC40001D8	RMI_PSMMU_DEACTIVATE
0xC40001DB	RMI_PSMMU_ST_L2_CREATE
0xC40001DC	RMI_PSMMU_ST_L2_DESTROY
0xC40001DD	RMI_DPT_L0_CREATE
0xC40001DE	RMI_DPT_L0_DESTROY
0xC40001DF	RMI_DPT_L1_CREATE
0xC40001E0	RMI_DPT_L1_DESTROY
0xC40001E1	RMI_GRANULE_TRACKING_GET
0xC40001E3	RMI_GRANULE_TRACKING_SET
0xC40001EC	RMI_RMM_CONFIG_GET
0xC40001EE	RMI_RMM_STATE_GET
0xC40001F0	RMI_PSMMU_EVENT_CONSUME
0xC40001F1	RMI_GRANULE_RANGE_DELEGATE
0xC40001F2	RMI_GRANULE_RANGE_UNDELEGATE
0xC40001F3	RMI_GPT_L1_CREATE
0xC40001F4	RMI_GPT_L1_DESTROY
0xC40001F5	RMI_RTT_DATA_MAP
0xC40001F6	RMI_RTT_DATA_UNMAP
0xC40001F7	RMI_RTT_DEV_MAP
0xC40001F8	RMI_RTT_DEV_UNMAP
0xC40001F9	RMI_RTT_ARCH_DEV_MAP

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

FID	Command
0xC40001FA	RMI_RTT_ARCH_DEV_UNMAP
0xC40001FB	RMI_RTT_UNPROT_MAP
0xC40001FC	RMI_RTT_UNPROT_UNMAP
0xC40001FD	RMI_RTT_AUX_PROT_MAP
0xC40001FE	RMI_RTT_AUX_PROT_UNMAP
0xC40001FF	RMI_RTT_AUX_UNPROT_MAP
0xC4000200	RMI_RTT_AUX_UNPROT_UNMAP
0xC4000201	RMI_REALM_TERMINATE
0xC4000202	RMI_RMM_ACTIVATE
0xC4000203	RMI_OP_CONTINUE
0xC4000204	RMI_PDEV_STREAM_CONNECT
0xC4000205	RMI_PDEV_STREAM_DISCONNECT
0xC4000206	RMI_PDEV_STREAM_COMPLETE
0xC4000207	RMI_PDEV_STREAM_KEY_PURGE
0xC4000208	RMI_OP_MEM_DONATE
0xC4000209	RMI_OP_MEM_RECLAIM
0xC400020A	RMI_OP_CANCEL
0xC400020B	RMI_VSMMU_FEATURES
0xC400020C	RMI_VSMMU_CMD_GET
0xC400020D	RMI_VSMMU_CMD_COMPLETE
0xC400020E	RMI_PSMMU_INFO

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.1 RMI_DPT_L0_CREATE command

Create a Level 0 DPT.

The RMI_DPT_L0_CREATE command may initiate a Stateful RMI Operation.

The RMI_DPT_L0_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- A9.7.5 *Device Permission Table*

B4.5.1.1 Interface

B4.5.1.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001DD`

B4.5.1.1.2 Context

The RMI_DPT_L0_CREATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
l0dpt	RmmDptL0	`DptL0()`	false	Level 0 DPT

DRAFT

B4.5.1.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.1.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_ats != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
l0dpt_state	`pre: l0dpt.state == DPT_L0_VALID` `post: result.status == RMI_ERROR_INPUT`

B4.5.1.2.1 Failure condition ordering

The RMI_DPT_L0_CREATE command does not have any failure condition orderings.

B4.5.1.3 Success conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
result	`post: result.status == RMI_SUCCESS`
state	`post: l0dpt.state == DPT_L0_VALID`

B4.5.1.4 Footprint

ID	Value
l0dpt_state	`l0dpt.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.2 RMI_DPT_L0_DESTROY command

Destroy a Level 0 DPT.

The RMI_DPT_L0_DESTROY command may initiate a Stateful RMI Operation.

The RMI_DPT_L0_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- A9.7.5 *Device Permission Table*

B4.5.2.1 Interface

B4.5.2.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001DE`

B4.5.2.1.2 Context

The RMI_DPT_L0_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
l0dpt	RmmDptL0	`DptL0()`	false	Level 0 DPT

B4.5.2.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

DRAFT

B4.5.2.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_ats != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
l0dpt_state	`pre: l0dpt.state == DPT_L0_INVALID` `post: result.status == RMI_ERROR_INPUT`

B4.5.2.2.1 Failure condition ordering

The RMI_DPT_L0_DESTROY command does not have any failure condition orderings.

B4.5.2.3 Success conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
result	`post: result.status == RMI_SUCCESS`
state	`post: l0dpt.state == DPT_L0_INVALID`

B4.5.2.4 Footprint

ID	Value
l0dpt_state	`l0dpt.state`
gran_state	`State of Granules in range [base, base + size)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.3 RMI_DPT_L1_CREATE command

Create a Level 1 DPT.

The RMI_DPT_L1_CREATE command may initiate a Stateful RMI Operation.

The RMI_DPT_L1_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- A9.7.5 *Device Permission Table*

B4.5.3.1 Interface

B4.5.3.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001DF`
addr	X1	63:0	Address	Base of physical address region described by the L1DPT

B4.5.3.1.2 Context

The RMI_DPT_L1_CREATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
l0dpt	RmmDptL0	`DptL0()`	false	Level 0 DPT
l0dpt_entry	RmmDptL0Entry	`DptL0Walk(addr)`	false	Level 0 DPT entry

DRAFT

B4.5.3.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.3.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_ats != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
l0dpt_state	`pre: l0dpt.state != DPT_L0_VALID` `post: result.status == RMI_ERROR_GLOBAL`
addr_bound	`pre: UInt(addr) >= rmm.static.dptps` `post: result.status == RMI_ERROR_INPUT`
addr_align	`pre: !AddrIsAligned(addr, rmm.static.l0dptsz)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
entry_state	`pre: l0dpt_entry.state == DPT_L0_ENTRY_TABLE` `post: result.status == RMI_ERROR_INPUT`

B4.5.3.2.1 Failure condition ordering

The RMI_DPT_L1_CREATE command does not have any failure condition orderings.

B4.5.3.3 Success conditions

ID	Condition
result	`post: result.status == RMI_SUCCESS`
state	`post: l0dpt_entry.state == DPT_L0_ENTRY_TABLE`

B4.5.3.4 Footprint

ID	Value
l0dpt_entry_state	`l0dpt_entry.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.4 RMI_DPT_L1_DESTROY command

Destroy a Level 1 DPT.

The RMI_DPT_L1_DESTROY command may initiate a Stateful RMI Operation.

The RMI_DPT_L1_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- A9.7.5 *Device Permission Table*

B4.5.4.1 Interface

B4.5.4.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001E0`
addr	X1	63:0	Address	Base of physical address region described by the L1DPT

B4.5.4.1.2 Context

The RMI_DPT_L1_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
l0dpt_entry	RmmDptL0Entry	`DptL0Walk(addr)`	false	Level 0 DPT entry

B4.5.4.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.4.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_ats != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
addr_bound	`pre: UInt(addr) >= rmm.static.dptps` `post: result.status == RMI_ERROR_INPUT`
addr_align	`pre: !AddrIsAligned(addr, rmm.static.l0dptsz)` `post: result.status == RMI_ERROR_INPUT`
entry_state	`pre: l0dpt_entry.state == DPT_L0_ENTRY_BLOCK` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.4.2.1 Failure condition ordering

The RMI_DPT_L1_DESTROY command does not have any failure condition orderings.

B4.5.4.3 Success conditions

ID	Condition
result	`post: result.status == RMI_SUCCESS`
state	`post: l0dpt_entry.state == DPT_L0_ENTRY_BLOCK`

B4.5.4.4 Footprint

ID	Value
l0dpt_entry_state	`l0dpt_entry.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.5 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
2	RMI feature register 2
3	RMI feature register 3
4	RMI feature register 4

See also:

- Chapter A3 *Feature discovery and configuration*

B4.5.5.1 Interface

B4.5.5.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000165`
index	X1	63:0	UInt64	Feature register index

B4.5.5.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
value	X1	63:0	Bits64	Feature register value

B4.5.5.2 Failure conditions

The RMI_FEATURES command does not have any failure conditions.

B4.5.5.3 Success conditions

ID	Condition
value	`post: value == RmiFeatureRegisterEncode(index)`

B4.5.5.4 Footprint

The RMI_FEATURES command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.6 RMI_GPT_L1_CREATE command

Create a Level 1 GPT.

The RMI_GPT_L1_CREATE command may initiate a Stateful RMI Operation.

The RMI_GPT_L1_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- A2.3.9 *Granule Protection Table management*
- B4.5.7 *RMI_GPT_L1_DESTROY command*

B4.5.6.1 Interface

B4.5.6.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F3`
addr	X1	63:0	Address	Base of physical address region described by the L1GPT

B4.5.6.1.2 Context

The RMI_GPT_L1_CREATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
l0gpt_entry	RmmGptL0Entry	`GptL0Walk(addr)`	false	Level 0 GPT entry

DRAFT

B4.5.6.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.6.2 Failure conditions

ID	Condition
addr_bound	`pre: UInt(addr) >= rmm.static.pasz` `post: result.status == RMI_ERROR_INPUT`
addr_align	`pre: !AddrIsAligned(addr, rmm.static.l0gptsz)` `post: result.status == RMI_ERROR_INPUT`
entry_state	`pre: l0gpt_entry.state == GPT_L0_ENTRY_TABLE` `post: result.status == RMI_ERROR_GPT`
unfold_denied	`pre: Unfolding of this L0GPT is denied.` `post: result.status == RMI_ERROR_GLOBAL`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
gran	pre: The RMM encountered a Granule within the memory donated for the L1GPT whose PAS is not NS or whose category is not DRAM. post: result.status == RMI_ERROR_INPUT

B4.5.6.2.1 Failure condition ordering

The RMI_GPT_L1_CREATE command does not have any failure condition orderings.

B4.5.6.3 Success conditions

ID	Condition
result	post: result.status == RMI_SUCCESS
state	post: l0gpt_entry.state == GPT_L0_ENTRY_TABLE

B4.5.6.4 Footprint

ID	Value
l0gpt_entry_state	l0gpt_entry.state
l0gpt_entry_addr	l0gpt_entry.addr

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.7 RMI_GPT_L1_DESTROY command

Destroy a Level 1 GPT.

The RMI_GPT_L1_DESTROY command may initiate a Stateful RMI Operation.

The RMI_GPT_L1_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- A2.3.9 *Granule Protection Table management*
- B4.5.6 *RMI_GPT_L1_CREATE command*

B4.5.7.1 Interface

B4.5.7.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F4`
addr	X1	63:0	Address	Base of physical address region described by the L1GPT

B4.5.7.1.2 Context

The RMI_GPT_L1_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
l0gpt_entry	RmmGptL0Entry	`GptL0Walk(addr)`	false	Level 0 GPT entry

DRAFT

B4.5.7.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.7.2 Failure conditions

ID	Condition
addr_bound	`pre: UInt(addr) >= rmm.static.pasz` `post: result.status == RMI_ERROR_INPUT`
addr_align	`pre: !AddrIsAligned(addr, rmm.static.l0gptsz)` `post: result.status == RMI_ERROR_INPUT`
entry_state	`pre: l0gpt_entry.state == GPT_L0_ENTRY_BLOCK` `post: result.status == RMI_ERROR_INPUT`
gpt_l1_homo	`pre: !GptL1IsHomogeneous(addr)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.7.2.1 Failure condition ordering

The RMI_GPT_L1_DESTROY command does not have any failure condition orderings.

B4.5.7.3 Success conditions

ID	Condition
result	`post: result.status == RMI_SUCCESS`
state	`post: l0gpt_entry.state == GPT_L0_ENTRY_BLOCK`

B4.5.7.4 Footprint

ID	Value
l0gpt_entry_state	`l0gpt_entry.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.8 RMI_GRANULE_RANGE_DELEGATE command

Delegates a range of Granules.

The RMI_GRANULE_RANGE_DELEGATE command may initiate a Stateful RMI Operation.

See also:

- A2.3.6.2 *Granule delegation*
- B4.3.5 *Range RMI operations*
- B4.5.9 *RMI_GRANULE_RANGE_UNDELEGATE command*

B4.5.8.1 Interface

B4.5.8.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F1`
base	X1	63:0	Address	Base PA of the target range
top	X2	63:0	Address	Top PA of the target range

B4.5.8.1.2 Context

The RMI_GRANULE_RANGE_DELEGATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state

DRAFT

B4.5.8.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top PA of range whose state is GRAN_DELEGATED

I_{DWGNJ} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.8.2 Failure conditions

ID	Condition
rmm_state	`pre: rmm.dynamic.state != RMM_STATE_ACTIVE` `post: result.status == RMI_ERROR_GLOBAL`
base_align	`pre: !AddrIsRmiGranuleAligned(base)` `post: result.status == RMI_ERROR_INPUT`
top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
top_bound	pre: UInt(top) <= UInt(base) post: result.status == RMI_ERROR_INPUT
populated	pre: While processing the target range, the RMM was unable to proceed due to memory being unpopulated. post: result.status == RMI_ERROR_INPUT
tracking	pre: While processing the target range, the RMM was unable to proceed due to the state of a tracking region. post: result.status == RMI_ERROR_TRACKING
state	pre: While processing the target range, the RMM encountered a Granule whose state is not GRAN_UNDELEGATED. post: result.status == RMI_ERROR_INPUT

B4.5.8.2.1 Failure condition ordering

The RMI_GRANULE_RANGE_DELEGATE command does not have any failure condition orderings.

B4.5.8.3 Success conditions

ID	Condition
state	post: GranulesAllState(base, out_top, GRAN_DELEGATED)
result	post: result.status == RMI_SUCCESS

B4.5.8.4 Footprint

ID	Value
gran_state	State of Granules in range [base, top)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.9 RMI_GRANULE_RANGE_UNDELEGATE command

Undelegates a range of Granules.

The RMI_GRANULE_RANGE_UNDELEGATE command may initiate a Stateful RMI Operation.

See also:

- A2.3.6.2 *Granule delegation*
- B4.3.5 *Range RMI operations*
- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*

B4.5.9.1 Interface

B4.5.9.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F2`
base	X1	63:0	Address	Base PA of the target range
top	X2	63:0	Address	Top PA of the target range

B4.5.9.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top PA of range whose state is GRAN_UNDELEGATED

DRAFT

I_{GYZHH} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.9.2 Failure conditions

ID	Condition
base_align	`pre: !AddrIsRmiGranuleAligned(base)` `post: result.status == RMI_ERROR_INPUT`
top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
top_bound	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
tracking	`pre: While processing the target range, the RMM was unable to proceed due to the state of a tracking region.` `post: result.status == RMI_ERROR_TRACKING`
state	`pre: While processing the target range, the RMM encountered a Granule whose state is not GRAN_DELEGATED.` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.9.2.1 Failure condition ordering

The RMI_GRANULE_RANGE_UNDELEGATE command does not have any failure condition orderings.

B4.5.9.3 Success conditions

ID	Condition
state	`post: GranulesAllState(base, out_top, GRAN_UNDELEGATED)`
content	`post: Contents of Granules in range [base, out_top) are wiped.`
result	`post: result.status == RMI_SUCCESS`

B4.5.9.4 Footprint

ID	Value
gran_state	`State of Granules in range [base, top)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.10 RMI_GRANULE_TRACKING_GET command

Get configuration of a Granule tracking region.

See also:

- A2.3.4 *Granule tracking region*
- B4.5.11 *RMI_GRANULE_TRACKING_SET command*

B4.5.10.1 Interface

B4.5.10.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001E1`
addr	X1	63:0	Address	PA of the tracking region

B4.5.10.1.2 Context

The RMI_GRANULE_TRACKING_GET command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
region	RmmTrackingRegion	`TrackingRegionAt(addr)`	false	Tracking region

B4.5.10.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
category	X1	1:0	RmiMemCategory	Memory category
state	X2	2:0	RmiTrackingRegionState	Tracking region state

The following unused bits of RMI_GRANULE_TRACKING_GET output values MBZ: X1[63:2], X2[63:3].

B4.5.10.2 Failure conditions

ID	Condition
addr_align	`pre: !AddrIsTrackingRegionAligned(addr)` `post: result.status == RMI_ERROR_INPUT`
addr_bound	`pre: UInt(addr) > rmm.static.pasz` `post: result.status == RMI_ERROR_INPUT`

B4.5.10.2.1 Failure condition ordering

The RMI_GRANULE_TRACKING_GET command does not have any failure condition orderings.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.10.3 Success conditions

ID	Condition
state	`post: Equal(state, region.state)`
category	`post: Equal(category, region.category)`

B4.5.10.4 Footprint

The RMI_GRANULE_TRACKING_GET command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.11 RMI_GRANULE_TRACKING_SET command

Set configuration of a Granule tracking region.

The RMI_GRANULE_TRACKING_SET command may initiate a Stateful RMI Operation.

The RMI_GRANULE_TRACKING_SET command may initiate a memory-transferring RMI Operation.

See also:

- A2.3.4 *Granule tracking region*
- B4.5.10 *RMI_GRANULE_TRACKING_GET command*

B4.5.11.1 Interface

B4.5.11.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001E3`
addr	X1	63:0	Address	PA of the tracking region
category	X2	1:0	RmiMemCategory	Memory category
state	X3	2:0	RmiTrackingRegionState	state

The following unused bits of RMI_GRANULE_TRACKING_SET input values SBZ: X2[63:2], X3[63:3].

B4.5.11.1.2 Context

The RMI_GRANULE_TRACKING_SET command operates on the following context.

Name	Type	Value	Before	Description
rmm_pre	RmmGlobal	`Rmm()`	true	RMM global state
rmm	RmmGlobal	`Rmm()`	false	RMM global state
region_pre	RmmTrackingRegion	`TrackingRegionAt(addr)`	true	Tracking region
region	RmmTrackingRegion	`TrackingRegionAt(addr)`	false	Tracking region

B4.5.11.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.11.2 Failure conditions

ID	Condition
rmm_state	`pre:  rmm.dynamic.state != RMM_STATE_ACTIVE` `post: result.status == RMI_ERROR_GLOBAL`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
state_valid	pre: (state != RMI_TRACKING_NONE && state != RMI_TRACKING_FINE && state != RMI_TRACKING_COARSE) post: result.status == RMI_ERROR_INPUT
addr_align	pre: !AddrIsTrackingRegionAligned(addr) post: result.status == RMI_ERROR_INPUT
addr_bound	pre: UInt(addr) > rmm.static.pasz post: result.status == RMI_ERROR_INPUT
category	pre: !MemCategoryIsCompatible(category, addr) post: result.status == RMI_ERROR_INPUT
reserved	pre: region.state == TRACKING_RESERVED post: result.status == RMI_ERROR_INPUT
in_state	pre: State of a Granule in input list is not GRAN_DELEGATED. post: result.status == RMI_ERROR_INPUT
in_loc_dram	pre: Tracking region is backed by DRAM and a Granule in input list is not located in DRAM. post: result.status == RMI_ERROR_INPUT

B4.5.11.2.1 Failure condition ordering

The RMI_GRANULE_TRACKING_SET command does not have any failure condition orderings.

B4.5.11.3 Success conditions

ID	Condition
tracked_inc	pre: (!TrackingRegionIsTracked(region_pre) && TrackingRegionIsTracked(region)) post: rmm.dynamic.num_tracked == rmm_pre.dynamic.num_tracked + 1
tracked_dec	pre: (TrackingRegionIsTracked(region_pre) && !TrackingRegionIsTracked(region)) post: rmm.dynamic.num_tracked == rmm_pre.dynamic.num_tracked - 1
result	post: result.status == RMI_SUCCESS
state	post: Equal(region.state, state)

B4.5.11.4 Footprint

ID	Value
num_tracked	rmm.dynamic.num_tracked

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.12 RMI_OP_CANCEL command

Cancel a stateful RMI operation.

See also:

- B4.3.2 *Stateful RMI operations*

B4.5.12.1 Interface

B4.5.12.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400020A`
handle	X1	63:0	Bits64	Handle which identifies the operation

B4.5.12.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.12.2 Failure conditions

ID	Condition
complete	`pre: !OperationIncomplete(handle)` `post: result.status == RMI_ERROR_INPUT`
can_cancel	`pre: OperationCanCancel(handle) != RMM_OP_CAN_CANCEL` `post: result.status == RMI_ERROR_INPUT`

B4.5.12.2.1 Failure condition ordering

The RMI_OP_CANCEL command does not have any failure condition orderings.

B4.5.12.3 Success conditions

The RMI_OP_CANCEL command does not have any success conditions.

B4.5.12.4 Footprint

The RMI_OP_CANCEL command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.13 RMI_OP_CONTINUE command

Continue a stateful RMI operation.

See also:

- B4.3.2 *Stateful RMI operations*

B4.5.13.1 Interface

B4.5.13.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000203`
handle	X1	63:0	Bits64	Handle which identifies the operation
flags	X2	63:0	RmiContinueFlags	Flags

B4.5.13.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
donate_req	X2	63:0	RmiOpMemDonateReq	Memory donation requirements. RES0 unless result.mem == RMI_OP_MEM_REQ_DONATE

DRAFT

B4.5.13.2 Failure conditions

ID	Condition
complete	`pre: !OperationIncomplete(handle)` `post: result.status == RMI_ERROR_INPUT`
fail	`pre: The resumed command fails.` `post: Output values are populated according to the specification of the resumed command.`

B4.5.13.2.1 Failure condition ordering

The RMI_OP_CONTINUE command does not have any failure condition orderings.

B4.5.13.3 Success conditions

ID	Condition
output_complete	`pre: !OperationIncomplete(handle)` `post: All output values are populated according to the specification of the completed command.`
status_incomplete	`pre: OperationIncomplete(handle)` `post: result.status == RMI_INCOMPLETE`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
donate_req	pre: (OperationIncomplete(handle) && result.data.incomplete.mem == RMI_OP_MEM_REQ_DONATE) post: donate_req describes the memory donation requirements of the operation.

B4.5.13.4 Footprint

The RMI_OP_CONTINUE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.14 RMI_OP_MEM_DONATE command

Donate memory to a stateful RMI operation.

See also:

- B4.3.2 *Stateful RMI operations*

B4.5.14.1 Interface

B4.5.14.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000208`
handle	X1	63:0	Bits64	Handle which identifies the operation
list_addr	X2	63:0	Address	PA of RMI Address Range List
list_count	X3	63:0	UInt64	Number of entries in RMI Address Range List

I_{PMFDY} The contents of the RMI Address Range List are not modified by execution of RMI_OP_MEM_DONATE.

B4.5.14.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
donated_count	X1	63:0	UInt64	Number of Granules consumed from RMI Address Range List
donate_req	X2	63:0	RmiOpMemDonateReq	Memory donation requirements RES0 unless result.status == RMI_INCOMPLETE and result.mem == RMI_OP_MEM_DONATE

DRAFT

B4.5.14.2 Failure conditions

ID	Condition
complete	`pre: !OperationIncomplete(handle)` `post: result.status == RMI_ERROR_INPUT`
list_align	`pre: !AddrIsAligned(list_addr, 8)` `post: result.status == RMI_ERROR_INPUT`
list_pas	`pre: !NonSecureAccessPermitted(list_addr)` `post: result.status == RMI_ERROR_INPUT`
mem_gt	`pre: Amount of memory described by RMI Address Range List is greater than amount required by the RMI operation.` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
mem_contig	pre: Memory described by RMI Address Range List does not meet contiguity requirement. post: result.status == RMI_ERROR_INPUT
mem_align	pre: Memory described by RMI Address Range List does not meet alignment requirement. post: result.status == RMI_ERROR_INPUT
mem_state	pre: State of a Granule described by RMI Address List does not match expected state. post: result.status == RMI_ERROR_INPUT

B4.5.14.2.1 Failure condition ordering

The RMI_OP_MEM_DONATE command does not have any failure condition orderings.

B4.5.14.3 Success conditions

ID	Condition
gran_state	post: State of the first donated_count Granules described by RMI Address Range List is GRAN_INTERNAL.

B4.5.14.4 Footprint

The RMI_OP_MEM_DONATE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.15 RMI_OP_MEM_RECLAIM command

Reclaim memory from a stateful RMI operation.

See also:

- B4.3.2 *Stateful RMI operations*

B4.5.15.1 Interface

B4.5.15.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000209`
handle	X1	63:0	Bits64	Handle which identifies the operation
list_addr	X2	63:0	Address	PA of RMI Address Range List
list_count	X3	63:0	UInt64	Number of entries in RMI Address Range List

B4.5.15.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
reclaim_count	X1	63:0	UInt64	Number of entries written to RMI Address Range List

DRAFT

B4.5.15.2 Failure conditions

ID	Condition
complete	`pre: !OperationIncomplete(handle)` `post: result.status == RMI_ERROR_INPUT`
list_align	`pre: !AddrIsAligned(list_addr, 8)` `post: result.status == RMI_ERROR_INPUT`
list_pas	`pre: !NonSecureAccessPermitted(list_addr)` `post: result.status == RMI_ERROR_INPUT`

B4.5.15.2.1 Failure condition ordering

The RMI_OP_MEM_RECLAIM command does not have any failure condition orderings.

B4.5.15.3 Success conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
gran_state	`post: State of Granules described by first reclaim_count entries in RMI Address Range List is the state described in each RMI Address Range Descriptor.`

B4.5.15.4 Footprint

The RMI_OP_MEM_RECLAIM command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.16 RMI_PDEV_ABORT command

Abort device communication associated with a PDEV.

See also:

- Chapter A9 *Realm device assignment*

B4.5.16.1 Interface

B4.5.16.1.1 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

B4.5.16.1.2 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

DRAFT

B4.5.16.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

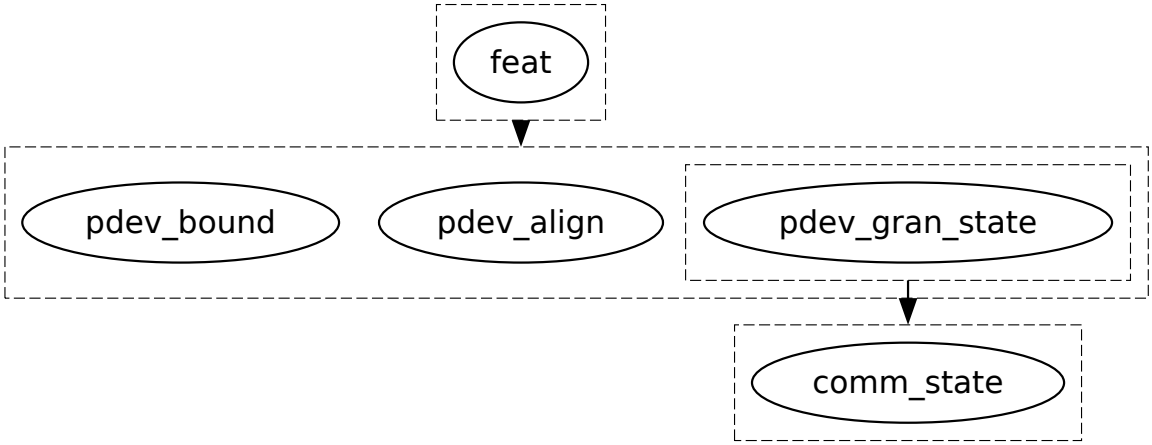
B4.5.16.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_bound	`pre: !PaIsTracked(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_gran_state	`pre: GranuleAt(pdev_ptr).state != GRAN_PDEV` `post: result.status == RMI_ERROR_INPUT`
comm_state	`pre: pdev.comm_state == DEV_COMM_IDLE` `post: result.status == RMI_ERROR_DEVICE`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.16.2.1 Failure condition ordering

```
[feat] < [pdev_align, pdev_bound, pdev_gran_state]
[pdev_gran_state] < [comm_state]
```



B4.5.16.3 Success conditions

ID	Condition
comm_state	`post: pdev.comm_state == DEV_COMM_IDLE`

B4.5.16.4 Footprint

ID	Value
comm_state	`pdev.comm_state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.17 RMI_PDEV_COMMUNICATE command

Perform device communication associated with a PDEV.

See also:

- Chapter A9 *Realm device assignment*

B4.5.17.1 Interface

B4.5.17.1.1 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

B4.5.17.1.2 Context

The RMI_PDEV_COMMUNICATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
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

DRAFT

B4.5.17.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.17.2 Failure conditions

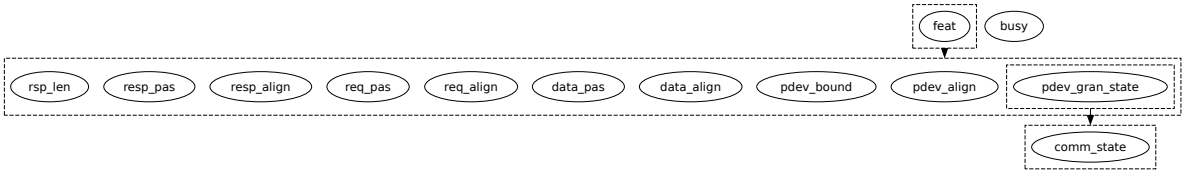
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_bound	`pre: !PaIsTracked(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_gran_state	`pre: GranuleAt(pdev_ptr).state != GRAN_PDEV` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
data_align	pre: !AddrIsRmiGranuleAligned(data_ptr) post: result.status == RMI_ERROR_INPUT
data_pas	pre: !NonSecureAccessPermitted(data_ptr) post: result.status == RMI_ERROR_INPUT
req_align	pre: !AddrIsRmiGranuleAligned(data.enter.req_addr) post: result.status == RMI_ERROR_INPUT
req_pas	pre: !NonSecureAccessPermitted(data.enter.req_addr) post: result.status == RMI_ERROR_INPUT
resp_align	pre: !AddrIsRmiGranuleAligned(data.enter.rsp_addr) post: result.status == RMI_ERROR_INPUT
resp_pas	pre: !NonSecureAccessPermitted(data.enter.rsp_addr) post: result.status == RMI_ERROR_INPUT
rsp_len	pre: data.enter.rsp_len > rmm.dynamic.rmi_granule_size post: result.status == RMI_ERROR_INPUT
comm_state	pre: (pdev.comm_state == DEV_COMM_IDLE || pdev.comm_state == DEV_COMM_ERROR) post: result.status == RMI_ERROR_DEVICE
busy	pre: PdevIsBusy(pdev) post: result.status == RMI_BUSY

B4.5.17.2.1 Failure condition ordering

```
[feat] < [pdev_align, pdev_bound, pdev_gran_state, data_align,
     data_pas, req_align, req_pas, resp_align, resp_pas, rsp_len]
[pdev_gran_state] < [comm_state]
```



B4.5.17.3 Success conditions

ID	Condition
comm_state	post: pdev.comm_state == DeviceCommunicate(pdev, data)
error	pre: DeviceCommunicate(pdev, data) == DEV_COMM_ERROR 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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
ready_no_spdm	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev_state_pre == PDEV_NEW && pdev.spdm == SPDM_FALSE) post: pdev.state == PDEV_READY
ready	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev_state_pre == PDEV_READY) post: pdev.state == PDEV_READY
stop_state	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev.op == PDEV_OP_STOP) post: pdev.state == PDEV_STOPPED
op_connect	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev.op == PDEV_OP_CONNECT) post: pdev.op == PDEV_OP_STREAM_COMPLETE
op_disconnect	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev.op == PDEV_OP_DISCONNECT) post: pdev.op == PDEV_OP_STREAM_COMPLETE
op_key_refresh	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev.op == PDEV_OP_KEY_REFRESH) post: pdev.op == PDEV_OP_STREAM_COMPLETE
op_none	pre: (DeviceCommunicate(pdev, data) == DEV_COMM_IDLE && pdev.op != PDEV_OP_CONNECT && pdev.op != PDEV_OP_DISCONNECT && pdev.op != PDEV_OP_KEY_REFRESH) post: pdev.op == PDEV_OP_NONE
complete	pre: DeviceCommunicate(pdev, data) == DEV_COMM_IDLE post: RmiDevCommComplete(data.exit.flags)
incomplete	pre: DeviceCommunicate(pdev, data) != DEV_COMM_IDLE post: !RmiDevCommComplete(data.exit.flags)

B4.5.17.4 Footprint

ID	Value
state	pdev.state
op	pdev_gran_statedev.op
comm_state	pdev.comm_state

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.18 RMI_PDEV_CREATE command

Create a PDEV.

The RMI_PDEV_CREATE command may initiate a Stateful RMI Operation.

The RMI_PDEV_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- Chapter A9 *Realm device assignment*
- A9.2.2 *Physical device invariants*
- B4.3.4 *Object creation and destruction*

B4.5.18.1 Interface

B4.5.18.1.1 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

B4.5.18.1.2 Context

The RMI_PDEV_CREATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
params	RmiPdevParams	`RmiPdevParamsAt(params_ptr)`	false	PDEV parameters

DRAFT

B4.5.18.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.18.2 Failure conditions

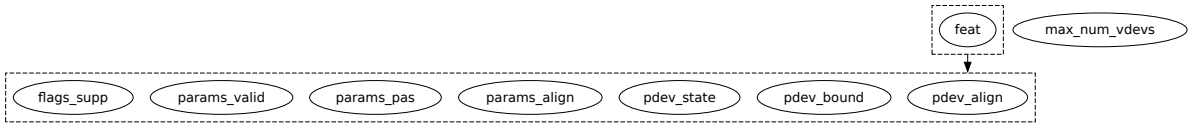
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_bound	`pre: !PaIsDelegableConventionalFine(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
pdev_state	`pre: GranuleAt(pdev_ptr).state != GRAN_DELEGATED` `post: result.status == RMI_ERROR_INPUT`
params_align	`pre: !AddrIsRmiGranuleAligned(params_ptr)` `post: result.status == RMI_ERROR_INPUT`
params_pas	`pre: !NonSecureAccessPermitted(params_ptr)` `post: result.status == RMI_ERROR_INPUT`
params_valid	`pre: !RmiPdevParamsIsValid(params_ptr)` `post: result.status == RMI_ERROR_INPUT`
flags_supp	`pre: !RmiPdevFlagsSupported(params.flags)` `post: result.status == RMI_ERROR_INPUT`
max_num_vdevs	`pre: params.max_vdevs_order > rmm.static.max_vdevs_order` `post: result.status == RMI_ERROR_INPUT`

B4.5.18.2.1 Failure condition ordering

```
[feat] < [pdev_align, pdev_bound, pdev_state, params_align,
     params_pas, params_valid, flags_supp]
```



B4.5.18.3 Success conditions

ID	Condition
gran_state	`post: GranuleAt(pdev_ptr).state == GRAN_PDEV`
category	`post: Equal(pdev.category, params.flags.category)`
pdev_id	`post: pdev.pdev_id == params.pdev_id`
routing_id	`post: pdev.routing_id == params.routing_id`
rid_base	`post: pdev.rid_base == params.rid_base`
rid_top	`post: pdev.rid_top == params.rid_top`
id_index	`post: pdev.id_index == params.id_index`
hash_algo	`post: Equal(pdev.hash_algo, params.hash_algo)`
spdm	`post: Equal(pdev.spdm, params.flags.spdm)`
state	`post: pdev.state == PDEV_NEW`
op	`post: pdev.op == PDEV_OP_NONE`
comm_state	`post: pdev.comm_state == DEV_COMM_PENDING`
max_num_vdevs	`post: pdev.max_num_vdevs == (2 ^ params.max_vdevs_order) - 1`
num_vdevs	`post: pdev.num_vdevs == 0`
pdev_stream_live	`post: !PdevStreamLive(pdev)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.18.4 Footprint

ID	Value
state	`GranuleAt(pdev_ptr).state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.19 RMI_PDEV_DESTROY command

Destroy a PDEV.

The RMI_PDEV_DESTROY command may initiate a Stateful RMI Operation.

The RMI_PDEV_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- Chapter A9 *Realm device assignment*
- B4.3.4 *Object creation and destruction*

B4.5.19.1 Interface

B4.5.19.1.1 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

B4.5.19.1.2 Context

The RMI_PDEV_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
pdev_pre	RmmPdev	`PdevAt(pdev_ptr)`	true	PDEV

B4.5.19.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.19.2 Failure conditions

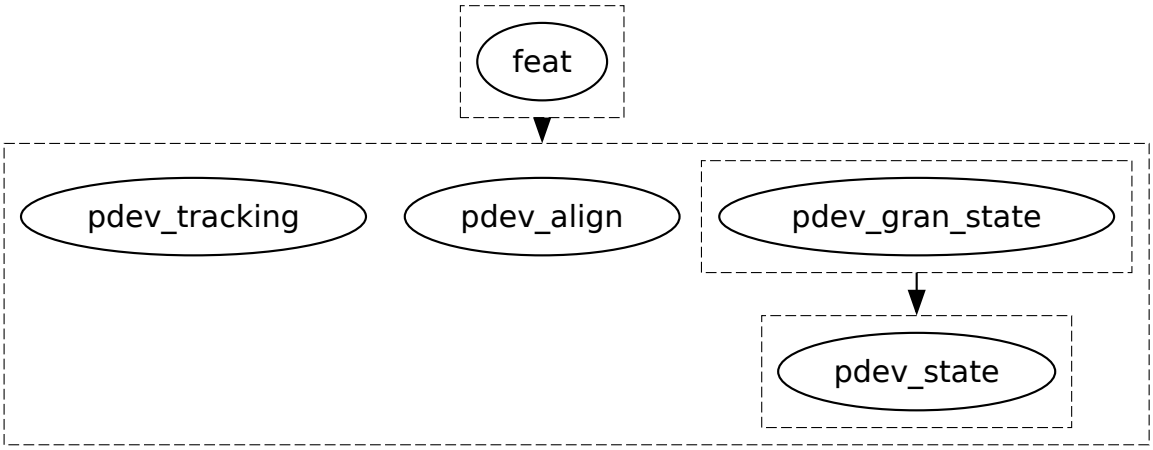
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_tracking	`pre: !PaIsTrackedFine(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_gran_state	`pre: GranuleAt(pdev_ptr).state != GRAN_PDEV` `post: result.status == RMI_ERROR_INPUT`
pdev_state	`pre: pdev_pre.state != PDEV_STOPPED` `post: result.status == RMI_ERROR_DEVICE`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.19.2.1 Failure condition ordering

```
[pdev_gran_state] < [pdev_state]
[feat] < [pdev_align, pdev_tracking, pdev_gran_state, pdev_state]
```



B4.5.19.3 Success conditions

ID	Condition
gran_state	`post: GranuleAt(pdev_ptr).state == GRAN_DELEGATED`

B4.5.19.4 Footprint

ID	Value
state	`GranuleAt(pdev_ptr).state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.20 RMI_PDEV_GET_STATE command

Get state of a PDEV.

See also:

- Chapter A9 *Realm device assignment*

B4.5.20.1 Interface

B4.5.20.1.1 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

B4.5.20.1.2 Context

The RMI_PDEV_GET_STATE command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV

B4.5.20.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
state	X1	7:0	RmiPdevState	PDEV state

The following unused bits of RMI_PDEV_GET_STATE output values MBZ: X1[63:8].

B4.5.20.2 Failure conditions

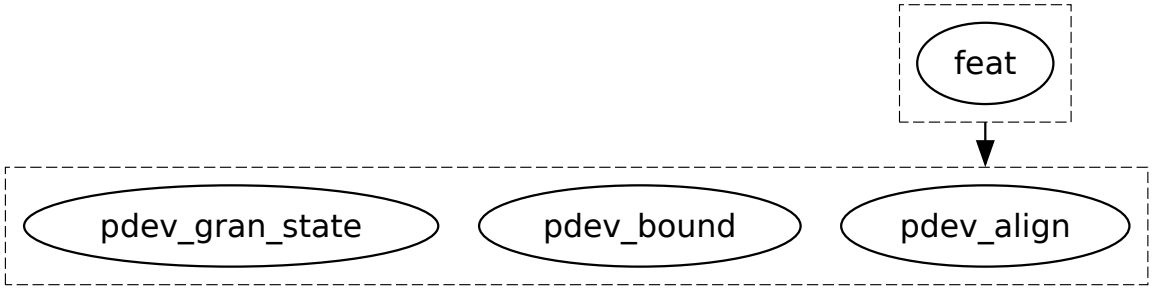
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_bound	`pre: !PaIsTracked(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_gran_state	`pre: GranuleAt(pdev_ptr).state != GRAN_PDEV` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.20.2.1 Failure condition ordering

```
[feat] < [pdev_align, pdev_bound, pdev_gran_state]
```



B4.5.20.3 Success conditions

ID	Condition
state	`post: Equal(state, pdev.state)`

B4.5.20.4 Footprint

The RMI_PDEV_GET_STATE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.21 RMI_PDEV_SET_PUBKEY command

Provide public key associated with a PDEV.

See also:

- Chapter A9 *Realm device assignment*

B4.5.21.1 Interface

B4.5.21.1.1 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

B4.5.21.1.2 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

DRAFT

B4.5.21.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.21.2 Failure conditions

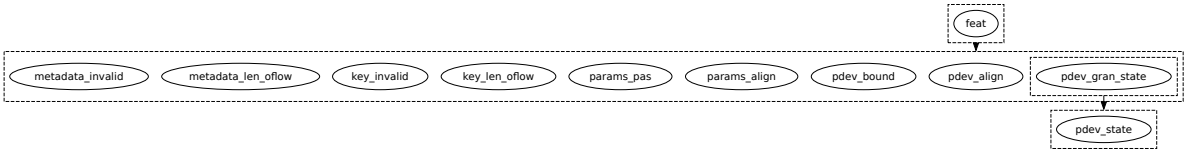
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_bound	`pre: !PaIsTracked(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_gran_state	`pre: GranuleAt(pdev_ptr).state != GRAN_PDEV` `post: result.status == RMI_ERROR_INPUT`
params_align	`pre: !AddrIsRmiGranuleAligned(params_ptr)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
params_pas	pre: !NonSecureAccessPermitted(params_ptr) post: result.status == RMI_ERROR_INPUT
key_len_oflow	pre: params.key_len > 1024 post: result.status == RMI_ERROR_INPUT
metadata_len_of low	pre: params.metadata_len > 1024 post: result.status == RMI_ERROR_INPUT
key_invalid	pre: Key is invalid, for example length is invalid for specified signature algorithm. post: result.status == RMI_ERROR_INPUT
metadata_invali d	pre: Metadata is invalid, for example length is invalid for specified signature algorithm. post: result.status == RMI_ERROR_INPUT
pdev_state	pre: pdev.state != PDEV_NEEDS_KEY post: result.status == RMI_ERROR_DEVICE

B4.5.21.2.1 Failure condition ordering

```
[feat] < [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]
```



B4.5.21.3 Success conditions

ID	Condition
state	post: pdev.state == PDEV_HAS_KEY
comm_state	post: pdev.comm_state == DEV_COMM_PENDING

B4.5.21.4 Footprint

ID	Value
state	pdev.state
comm_state	pdev.comm_state

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.22 RMI_PDEV_STOP command

Stop a PDEV.

See also:

- Chapter A9 *Realm device assignment*

B4.5.22.1 Interface

B4.5.22.1.1 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

B4.5.22.1.2 Context

The RMI_PDEV_STOP command operates on the following context.

Name	Type	Value	Before	Description
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV

B4.5.22.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.22.2 Failure conditions

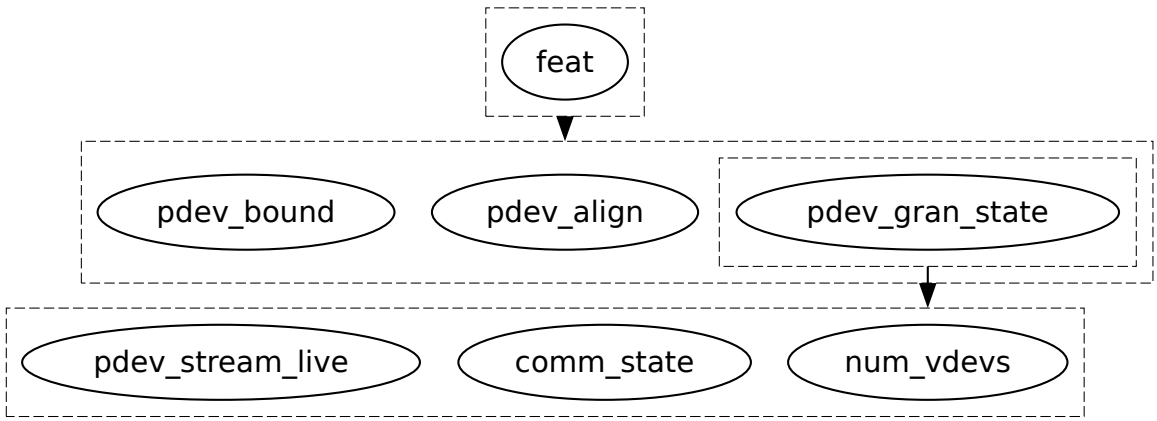
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_bound	`pre: !PaIsTracked(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_gran_state	`pre: GranuleAt(pdev_ptr).state != GRAN_PDEV` `post: result.status == RMI_ERROR_INPUT`
comm_state	`pre: pdev.comm_state != DEV_COMM_IDLE` `post: result.status == RMI_ERROR_DEVICE`
num_vdevs	`pre: pdev.num_vdevs != 0` `post: result.status == RMI_ERROR_DEVICE`
pdev_stream_liv e	`pre: PdevStreamLive(pdev)` `post: result.status == RMI_ERROR_DEVICE`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.22.2.1 Failure condition ordering

```
[feat] < [pdev_align, pdev_bound, pdev_gran_state]
[pdev_gran_state] < [num_vdevs, comm_state, pdev_stream_live]
```



B4.5.22.3 Success conditions

ID	Condition
op	`post: pdev.op == PDEV_OP_STOP`
comm_state	`post: pdev.comm_state == DEV_COMM_PENDING`

B4.5.22.4 Footprint

ID	Value
op	`pdev.op`
comm_state	`pdev.comm_state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.23 RMI_PDEV_STREAM_COMPLETE command

Complete an operation on a PDEV stream.

See also:

- A9.3 *Physical device stream object*

B4.5.23.1 Interface

B4.5.23.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000206`
pdev_1_ptr	X1	63:0	Address	PA of the first PDEV object
pdev_2_ptr	X2	63:0	Address	PA of the second PDEV object
stream_hnd	X3	63:0	Bits64	Stream handle

B4.5.23.1.2 Context

The RMI_PDEV_STREAM_COMPLETE command operates on the following context.

Name	Type	Value	Before	Description
pdev_1	RmmPdev	`PdevAt(pdev_1_ptr)`	false	First PDEV object
pdev_2	RmmPdev	`PdevAt(pdev_2_ptr)`	false	Second PDEV object
stream_result	RmmPdevStreamResult	`PdevStreamFromHandle(pdev_1, pdev_2, stream_hnd)`	false	Result of looking up PDEV stream
stream	RmmPdevStream	`stream_result.stream`	false	PDEV stream

B4.5.23.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.23.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
stream_valid	`pre: stream_result.valid != RMM_TRUE` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
stream_state	pre: !(stream.state IN { PDEV_STREAM_CONNECTING, PDEV_STREAM_DISCONNECTING, PDEV_STREAM_KEY_REFRESHING, PDEV_STREAM_KEY_PURGING }) post: result.status == RMI_ERROR_DEVICE
pdev_1_align	pre: !AddrIsRmiGranuleAligned(pdev_1_ptr) post: result.status == RMI_ERROR_INPUT
pdev_1_bound	pre: !PaIsTracked(pdev_1_ptr) post: result.status == RMI_ERROR_INPUT
pdev_1_gran_sta te	pre: GranuleAt(pdev_1_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
pdev_1_state	pre: pdev_1.state != PDEV_READY post: result.status == RMI_ERROR_INPUT
pdev_1_comm_sta te	pre: pdev_1.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_INPUT
pdev_2_align	pre: (PdevStreamPdev2Required(stream.stream_type) && !AddrIsRmiGranuleAligned(pdev_2_ptr)) post: result.status == RMI_ERROR_INPUT
pdev_2_bound	pre: (PdevStreamPdev2Required(stream.stream_type) && !PaIsTracked(pdev_2_ptr)) post: result.status == RMI_ERROR_INPUT
pdev_2_gran_sta te	pre: (PdevStreamPdev2Required(stream.stream_type) && GranuleAt(pdev_2_ptr).state != GRAN_PDEV) post: result.status == RMI_ERROR_INPUT
pdev_2_state	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.state != PDEV_READY) post: result.status == RMI_ERROR_INPUT
pdev_2_comm_sta te	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.comm_state != DEV_COMM_IDLE) post: result.status == RMI_ERROR_INPUT
complete	pre: (pdev_1.op != PDEV_OP_STREAM_COMPLETE) || (PdevStreamPdev2Required(stream.stream_type) && (pdev_2.op != PDEV_OP_STREAM_COMPLETE)) post: result.status == RMI_ERROR_DEVICE

B4.5.23.2.1 Failure condition ordering

The RMI_PDEV_STREAM_COMPLETE command does not have any failure condition orderings.

B4.5.23.3 Success conditions

ID	Condition
connected	pre: stream.state == PDEV_STREAM_CONNECTING post: stream.state == PDEV_STREAM_CONNECTED
disconnected	pre: stream.state == PDEV_STREAM_DISCONNECTING post: stream.state == PDEV_STREAM_DISCONNECTED

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
key_refreshed	pre: stream.state == PDEV_STREAM_KEY_REFRESHING post: stream.state == PDEV_STREAM_CONNECTED
key_purged	pre: stream.state == PDEV_STREAM_KEY_PURGING post: stream.state == PDEV_STREAM_CONNECTED
pdev_1_op	post: pdev_1.op == PDEV_OP_NONE
pdev_2_op	pre: PdevStreamPdev2Required(stream.stream_type) post: pdev_2.op == PDEV_OP_NONE

B4.5.23.4 Footprint

The RMI_PDEV_STREAM_COMPLETE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.24 RMI_PDEV_STREAM_CONNECT command

Initiate connection of a PDEV stream.

See also:

- A9.3 *Physical device stream object*

B4.5.24.1 Interface

B4.5.24.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000204`
params_ptr	X1	63:0	Address	PA of PDEV stream parameters

B4.5.24.1.2 Context

The RMI_PDEV_STREAM_CONNECT command operates on the following context.

Name	Type	Value	Before	Description
params	RmiPdevStreamParams	`RmiPdevStreamParamsAt(params_ptr)`	false	Parameters
pdev_1	RmmPdev	`PdevAt(params.pdev_1)`	false	First PDEV object
pdev_2	RmmPdev	`PdevAt(params.pdev_2)`	false	Second PDEV object
stream_result	RmmPdevStreamResult	`PdevStreamAlloc(pdev_1, pdev_2)`	false	Result of allocating PDEV stream
stream	RmmPdevStream	`stream_result.stream`	false	PDEV stream

DRAFT

B4.5.24.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
stream_hnd	X1	63:0	Bits64	PDEV stream handle

B4.5.24.2 Failure conditions

ID	Condition
feat_da	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
params_align	`pre: !AddrIsRmiGranuleAligned(params_ptr)` `post: result.status == RMI_ERROR_INPUT`
params_pas	`pre: !NonSecureAccessPermitted(params_ptr)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
params_valid	pre: !RmiPdevStreamParamsIsValid(params_ptr) post: result.status == RMI_ERROR_INPUT
feat_non_tee_st ream	pre: (params.stream_type == RMI_PDEV_STREAM_NON_TEE && Rmm().static.feat_non_tee_stream != FEATURE_TRUE) post: result.status == RMI_ERROR_NOT_SUPPORTED
pdev_1_align	pre: !AddrIsRmiGranuleAligned(params.pdev_1) post: result.status == RMI_ERROR_INPUT
pdev_1_bound	pre: !PaIsTracked(params.pdev_1) post: result.status == RMI_ERROR_INPUT
pdev_1_gran_sta te	pre: GranuleAt(params.pdev_1).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT

pdev_1_category	pre: !pdev_1.category IN {
	PDEV_ENDPOINT_ACCEL_OFF_CHIP,
	PDEV_ENDPOINT_ACCEL_ON_CHIP
	}
	post: result.status == RMI_ERROR_INPUT

pdev_1_state	pre: pdev_1.state != PDEV_READY post: result.status == RMI_ERROR_INPUT
pdev_1_op	pre: pdev_1.op != PDEV_OP_NONE post: result.status == RMI_ERROR_INPUT
pdev_1_comm_sta te	pre: pdev_1.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_INPUT
pdev_2_align	pre: (PdevStreamPdev2Required(params.stream_type) && !AddrIsRmiGranuleAligned(params.pdev_2)) post: result.status == RMI_ERROR_INPUT
pdev_2_bound	pre: (PdevStreamPdev2Required(params.stream_type) && !PaIsTracked(params.pdev_2)) post: result.status == RMI_ERROR_INPUT
pdev_2_gran_sta te	pre: (PdevStreamPdev2Required(params.stream_type) && GranuleAt(params.pdev_2).state != GRAN_PDEV) post: result.status == RMI_ERROR_INPUT
pdev_2_category	pre: (PdevStreamPdev2Required(params.stream_type) && pdev_2.category != PdevStreamPdev2Category(params.stream_type)) post: result.status == RMI_ERROR_INPUT
pdev_2_state	pre: (PdevStreamPdev2Required(params.stream_type) && pdev_2.state != PDEV_READY) post: result.status == RMI_ERROR_INPUT
pdev_2_op	pre: (PdevStreamPdev2Required(params.stream_type) && pdev_2.op != PDEV_OP_NONE) post: result.status == RMI_ERROR_INPUT
pdev_2_comm_sta te	pre: (PdevStreamPdev2Required(params.stream_type) && pdev_2.comm_state != DEV_COMM_IDLE) post: result.status == RMI_ERROR_INPUT
disconnected	pre: stream.state != PDEV_STREAM_DISCONNECTED post: result.status == RMI_ERROR_INPUT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
stream_type_exi sts	pre: PdevStreamFromType(pdev_1, PdevStreamTypeFromRmi(params.stream_type)).valid == RMM_TRUE post: result.status == RMI_ERROR_INPUT
stream_alloc_fa iled	pre: stream_result.valid != RMM_TRUE post: result.status == RMI_ERROR_INPUT

B4.5.24.2.1 Failure condition ordering

The RMI_PDEV_STREAM_CONNECT command does not have any failure condition orderings.

B4.5.24.3 Success conditions

ID	Condition
stream_hnd	post: stream_hnd == stream.handle
pdev_1_op	post: pdev_1.op == PDEV_OP_CONNECT
pdev_1_comm_state	post: pdev_1.comm_state == DEV_COMM_PENDING
pdev_2_op	pre: PdevStreamPdev2Required(params.stream_type) post: pdev_2.op == PDEV_OP_CONNECT
pdev_2_comm_state	pre: PdevStreamPdev2Required(params.stream_type) post: pdev_2.comm_state == DEV_COMM_PENDING
state	post: stream.state == PDEV_STREAM_CONNECTING
stream_type	post: Equal(stream.stream_type, params.stream_type)
ide_sid	post: stream.ide_sid == params.ide_sid
num_addr_range	post: stream.num_addr_range == params.num_addr_range
addr_range	post: RmiAddrRangesEqual16(stream.addr_range, params.addr_range, params.num_addr_range)

B4.5.24.4 Footprint

The RMI_PDEV_STREAM_CONNECT command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.25 RMI_PDEV_STREAM_DISCONNECT command

Initiate disconnection of a PDEV stream.

See also:

- A9.3 *Physical device stream object*

B4.5.25.1 Interface

B4.5.25.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000205`
pdev_1_ptr	X1	63:0	Address	PA of the first PDEV object
pdev_2_ptr	X2	63:0	Address	PA of the second PDEV object
stream_hnd	X3	63:0	Bits64	Stream handle

B4.5.25.1.2 Context

The RMI_PDEV_STREAM_DISCONNECT command operates on the following context.

Name	Type	Value	Before	Description
pdev_1	RmmPdev	`PdevAt(pdev_1_ptr)`	false	First PDEV object
pdev_2	RmmPdev	`PdevAt(pdev_2_ptr)`	false	Second PDEV object
stream_result	RmmPdevStreamResult	`PdevStreamFromHandle(pdev_1, pdev_2, stream_hnd)`	false	Result of looking up PDEV stream
stream	RmmPdevStream	`stream_result.stream`	false	PDEV stream

B4.5.25.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.25.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
pdev_1_align	`pre: !AddrIsRmiGranuleAligned(pdev_1_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_1_bound	`pre: !PaIsTracked(pdev_1_ptr)` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
pdev_1_gran_sta te	pre: GranuleAt(pdev_1_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
pdev_1_state	pre: (pdev_1.state != PDEV_READY && pdev_1.state != PDEV_ERROR) post: result.status == RMI_ERROR_INPUT
pdev_1_comm_sta te	pre: pdev_1.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_INPUT
pdev_1_num_vdev s	pre: pdev_1.num_vdevs != 0 post: result.status == RMI_ERROR_DEVICE
pdev_2_align	pre: (PdevStreamPdev2Required(stream.stream_type) && !AddrIsRmiGranuleAligned(pdev_2_ptr)) post: result.status == RMI_ERROR_INPUT
pdev_2_bound	pre: (PdevStreamPdev2Required(stream.stream_type) && !PaIsTracked(pdev_2_ptr)) post: result.status == RMI_ERROR_INPUT
pdev_2_gran_sta te	pre: (PdevStreamPdev2Required(stream.stream_type) && GranuleAt(pdev_2_ptr).state != GRAN_PDEV) post: result.status == RMI_ERROR_INPUT
pdev_2_state	pre: (PdevStreamPdev2Required(stream.stream_type) && (pdev_2.state != PDEV_READY && pdev_2.state != PDEV_ERROR)) post: result.status == RMI_ERROR_INPUT
pdev_2_comm_sta te	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.comm_state != DEV_COMM_IDLE) post: result.status == RMI_ERROR_INPUT
stream_valid	pre: stream_result.valid != RMM_TRUE post: result.status == RMI_ERROR_INPUT
stream_state	pre: stream.state != PDEV_STREAM_CONNECTED post: result.status == RMI_ERROR_DEVICE

B4.5.25.2.1 Failure condition ordering

The RMI_PDEV_STREAM_DISCONNECT command does not have any failure condition orderings.

B4.5.25.3 Success conditions

ID	Condition
state	post: stream.state == PDEV_STREAM_DISCONNECTING
pdev_1_op	post: pdev_1.op == PDEV_OP_DISCONNECT
pdev_1_comm_state	post: pdev_1.comm_state == DEV_COMM_PENDING
pdev_2_op	pre: PdevStreamPdev2Required(stream.stream_type) post: pdev_2.op == PDEV_OP_DISCONNECT
pdev_2_comm_state	pre: PdevStreamPdev2Required(stream.stream_type) post: pdev_2.comm_state == DEV_COMM_PENDING

B4.5.25.4 Footprint

The RMI_PDEV_STREAM_DISCONNECT command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.26 RMI_PDEV_STREAM_KEY_PURGE command

Initiate purge of inactive keys from a PDEV stream.

See also:

- A9.3 *Physical device stream object*

B4.5.26.1 Interface

B4.5.26.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000207`
pdev_1_ptr	X1	63:0	Address	PA of the first PDEV object
pdev_2_ptr	X2	63:0	Address	PA of the second PDEV object
stream_hnd	X3	63:0	Bits64	Stream handle

B4.5.26.1.2 Context

The RMI_PDEV_STREAM_KEY_PURGE command operates on the following context.

Name	Type	Value	Before	Description
pdev_1	RmmPdev	`PdevAt(pdev_1_ptr)`	false	First PDEV object
pdev_2	RmmPdev	`PdevAt(pdev_2_ptr)`	false	Second PDEV object

stream_result	RmmPdevStreamResult	`PdevStreamFromHandle(`
		`  pdev_1, pdev_2,`
		`  stream_hnd)`
			false	Result of looking up PDEV stream

stream	RmmPdevStream	`stream_result.stream`	false	PDEV stream

B4.5.26.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.26.2 Failure conditions

ID	Condition
feat	`pre:  Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
stream_valid	`pre:  stream_result.valid != RMM_TRUE` `post: result.status == RMI_ERROR_INPUT`
stream_state	`pre:  stream.state != PDEV_STREAM_CONNECTED` `post: result.status == RMI_ERROR_DEVICE`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
stream_type	pre: !stream.stream_type IN { PDEV_STREAM_NCOH_SYS, PDEV_STREAM_COH_SYS } post: result.status == RMI_ERROR_DEVICE
pdev_1_align	pre: !AddrIsRmiGranuleAligned(pdev_1_ptr) post: result.status == RMI_ERROR_INPUT
pdev_1_bound	pre: !PaIsTracked(pdev_1_ptr) post: result.status == RMI_ERROR_INPUT
pdev_1_gran_sta te	pre: GranuleAt(pdev_1_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
pdev_1_state	pre: pdev_1.state != PDEV_READY post: result.status == RMI_ERROR_INPUT
pdev_1_op	pre: pdev_1.op != PDEV_OP_NONE post: result.status == RMI_ERROR_INPUT
pdev_1_comm_sta te	pre: pdev_1.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_INPUT
pdev_2_align	pre: (PdevStreamPdev2Required(stream.stream_type) && !AddrIsRmiGranuleAligned(pdev_2_ptr)) post: result.status == RMI_ERROR_INPUT
pdev_2_bound	pre: (PdevStreamPdev2Required(stream.stream_type) && !PaIsTracked(pdev_2_ptr)) post: result.status == RMI_ERROR_INPUT
pdev_2_gran_sta te	pre: (PdevStreamPdev2Required(stream.stream_type) && GranuleAt(pdev_2_ptr).state != GRAN_PDEV) post: result.status == RMI_ERROR_INPUT
pdev_2_state	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.state != PDEV_READY) post: result.status == RMI_ERROR_INPUT
pdev_2_op	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.op != PDEV_OP_NONE) post: result.status == RMI_ERROR_INPUT
pdev_2_comm_sta te	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.comm_state != DEV_COMM_IDLE) post: result.status == RMI_ERROR_INPUT

B4.5.26.2.1 Failure condition ordering

The RMI_PDEV_STREAM_KEY_PURGE command does not have any failure condition orderings.

B4.5.26.3 Success conditions

ID	Condition
state	post: stream.state == PDEV_STREAM_KEY_PURGING
pdev_1_op	post: pdev_1.op == PDEV_OP_KEY_PURGE
pdev_1_comm_state	post: pdev_1.comm_state == DEV_COMM_PENDING
pdev_2_op	pre: PdevStreamPdev2Required(stream.stream_type) post: pdev_2.op == PDEV_OP_KEY_PURGE

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
pdev_2_comm_state	`pre: PdevStreamPdev2Required(stream.stream_type)` `post: pdev_2.comm_state == DEV_COMM_PENDING`

B4.5.26.4 Footprint

The RMI_PDEV_STREAM_KEY_PURGE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.27 RMI_PDEV_STREAM_KEY_REFRESH command

Initiate key refresh of a PDEV stream.

See also:

- A9.3 *Physical device stream object*

B4.5.27.1 Interface

B4.5.27.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400017A`
pdev_1_ptr	X1	63:0	Address	PA of the first PDEV object
pdev_2_ptr	X2	63:0	Address	PA of the second PDEV object
stream_hnd	X3	63:0	Bits64	Stream handle

B4.5.27.1.2 Context

The RMI_PDEV_STREAM_KEY_REFRESH command operates on the following context.

Name	Type	Value	Before	Description
pdev_1	RmmPdev	`PdevAt(pdev_1_ptr)`	false	First PDEV object
pdev_2	RmmPdev	`PdevAt(pdev_2_ptr)`	false	Second PDEV object
stream_result	RmmPdevStreamResult	`PdevStreamFromHandle(pdev_1, pdev_2, stream_hnd)`	false	Result of looking up PDEV stream
stream	RmmPdevStream	`stream_result.stream`	false	PDEV stream

B4.5.27.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.27.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
stream_valid	`pre: stream_result.valid != RMM_TRUE` `post: result.status == RMI_ERROR_INPUT`

stream_type	`pre: (stream.stream_type == PDEV_STREAM_NCOH_SYS`
	`|| stream.stream_type == PDEV_STREAM_COH_SYS)`
	`post: result.status == RMI_ERROR_DEVICE`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
stream_state	pre: stream.state != PDEV_STREAM_CONNECTED post: result.status == RMI_ERROR_DEVICE
pdev_1_align	pre: !AddrIsRmiGranuleAligned(pdev_1_ptr) post: result.status == RMI_ERROR_INPUT
pdev_1_bound	pre: !PaIsTracked(pdev_1_ptr) post: result.status == RMI_ERROR_INPUT
pdev_1_gran_sta te	pre: GranuleAt(pdev_1_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
pdev_1_state	pre: pdev_1.state != PDEV_READY post: result.status == RMI_ERROR_INPUT
pdev_1_op	pre: pdev_1.op != PDEV_OP_NONE post: result.status == RMI_ERROR_INPUT
pdev_1_comm_sta te	pre: pdev_1.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_INPUT
pdev_2_align	pre: (PdevStreamPdev2Required(stream.stream_type) && !AddrIsRmiGranuleAligned(pdev_2_ptr)) post: result.status == RMI_ERROR_INPUT
pdev_2_bound	pre: (PdevStreamPdev2Required(stream.stream_type) && !PaIsTracked(pdev_2_ptr)) post: result.status == RMI_ERROR_INPUT
pdev_2_gran_sta te	pre: (PdevStreamPdev2Required(stream.stream_type) && GranuleAt(pdev_2_ptr).state != GRAN_PDEV) post: result.status == RMI_ERROR_INPUT
pdev_2_state	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.state != PDEV_READY) post: result.status == RMI_ERROR_INPUT
pdev_2_op	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.op != PDEV_OP_NONE) post: result.status == RMI_ERROR_INPUT
pdev_2_comm_sta te	pre: (PdevStreamPdev2Required(stream.stream_type) && pdev_2.comm_state != DEV_COMM_IDLE) post: result.status == RMI_ERROR_INPUT

B4.5.27.2.1 Failure condition ordering

The RMI_PDEV_STREAM_KEY_REFRESH command does not have any failure condition orderings.

B4.5.27.3 Success conditions

ID	Condition
state	post: stream.state == PDEV_STREAM_KEY_REFRESHING
pdev_1_op	post: pdev_1.op == PDEV_OP_KEY_REFRESH
pdev_1_comm_state	post: pdev_1.comm_state == DEV_COMM_PENDING
pdev_2_op	pre: PdevStreamPdev2Required(stream.stream_type) post: pdev_2.op == PDEV_OP_KEY_REFRESH
pdev_2_comm_state	pre: PdevStreamPdev2Required(stream.stream_type) post: pdev_2.comm_state == DEV_COMM_PENDING

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.27.4 Footprint

The RMI_PDEV_STREAM_KEY_REFRESH command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.28 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*

B4.5.28.1 Interface

B4.5.28.1.1 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

B4.5.28.1.2 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

B4.5.28.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.28.2 Failure conditions

ID	Condition
alias	`pre: calling_rec_ptr == target_rec_ptr` `post: result.status == RMI_ERROR_INPUT`
calling_align	`pre: !AddrIsRmiGranuleAligned(calling_rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
calling_bound	`pre: !PaIsTracked(calling_rec_ptr)` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
calling_state	pre: GranuleAt(calling_rec_ptr).state != GRAN_REC post: result.status == RMI_ERROR_INPUT
target_align	pre: !AddrIsRmiGranuleAligned(target_rec_ptr) post: result.status == RMI_ERROR_INPUT
target_bound	pre: !PaIsTracked(target_rec_ptr) post: result.status == RMI_ERROR_INPUT
target_state	pre: GranuleAt(target_rec_ptr).state != GRAN_REC post: result.status == RMI_ERROR_INPUT
pending	pre: calling_rec.pending != REC_PENDING_PSCI post: result.status == RMI_ERROR_INPUT
owner	pre: target_rec.owner != calling_rec.owner post: result.status == RMI_ERROR_INPUT
target	pre: target_rec.mpidr != calling_rec.gprs[[1]] post: result.status == RMI_ERROR_INPUT
status	pre: !PsciReturnCodePermitted(calling_rec, target_rec, status) post: result.status == RMI_ERROR_INPUT

B4.5.28.2.1 Failure condition ordering

The RMI_PSCI_COMPLETE command does not have any failure condition orderings.

B4.5.28.3 Success conditions

ID	Condition
pending	post: calling_rec.pending == REC_PENDING_NONE
on_already	pre: (status == PSCI_SUCCESS && calling_rec.gprs[[0]] == FID_PSCI_CPU_ON && target_rec.flags.runnable == RUNNABLE) post: (calling_rec.gprs[[0]] == PsciReturnCodeEncode(PSCI_ALREADY_ON))

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
on_success	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))
affinity_on	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))
affinity_off	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))
status	pre: status != PSCI_SUCCESS post: (calling_rec.gprs[[0]] == PsciReturnCodeEncode(status))
args	post: (calling_rec.gprs[[1]] == Zeros{64}() && calling_rec.gprs[[2]] == Zeros{64}() && calling_rec.gprs[[3]] == Zeros{64}())

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.28.4 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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.29 RMI_PSMMU_ACTIVATE command

Activate a PSMMU.

The RMI_PSMMU_ACTIVATE command may initiate a Stateful RMI Operation.

The RMI_PSMMU_ACTIVATE command may initiate a memory-transferring RMI Operation.

See also:

- A9.7 *Physical SMMU*
- A9.7.6 *PSMMU interrupts*
- B4.5.30 *RMI_PSMMU_DEACTIVATE command*

B4.5.29.1 Interface

B4.5.29.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001D7`
psmmu_ptr	X1	63:0	Address	PA of PSMMU
params_ptr	X2	63:0	Address	PA of PSMMU parameters

B4.5.29.1.2 Context

The RMI_PSMMU_ACTIVATE command operates on the following context.

Name	Type	Value	Before	Description
psmmu	RmmPsmmu	`PsmmuAt(psmmu_ptr)`	false	PSMMU
params	RmiPsmmuParams	`RmiPsmmuParamsAt(params_ptr)`	false	PSMMU parameters

B4.5.29.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.29.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
psmmu_valid	`pre: !PsmmuAddrIsValid(psmmu_ptr)` `post: result.status == RMI_ERROR_INPUT`
psmmu_state	`pre: psmmu.state != PSMMU_INACTIVE` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
params_align	pre: !AddrIsRmiGranuleAligned(params_ptr) post: result.status == RMI_ERROR_INPUT
params_pas	pre: !NonSecureAccessPermitted(params_ptr) post: result.status == RMI_ERROR_INPUT
irq_cfg	pre: Equal(params.flags.irq_cfg, psmmu.irq_cfg) post: result.status == RMI_ERROR_INPUT
cmdq_sync_irq_w ired	pre: Equal(params.flags.cmdq_sync_irq_wired, psmmu.cmdq_sync_irq_wired ↪) post: result.status == RMI_ERROR_INPUT
gerror_valid	pre: (params.flags.irq_cfg == RMI_IRQ_MSI && !MsiAddrIsValid(params.gerr_addr)) post: result.status == RMI_ERROR_INPUT
eventq_valid	pre: (params.flags.irq_cfg == RMI_IRQ_MSI && !MsiAddrIsValid(params.eventq_addr)) post: result.status == RMI_ERROR_INPUT
priq_valid	pre: (params.flags.irq_cfg == RMI_IRQ_MSI && !MsiAddrIsValid(params.priq_addr)) post: result.status == RMI_ERROR_INPUT
ats_supp	pre: (params.flags.ats == RMI_FEATURE_TRUE && psmmu.feat_ats != FEATURE_TRUE) post: result.status == RMI_ERROR_INPUT
pri_supp	pre: (params.flags.pri == RMI_FEATURE_TRUE && psmmu.feat_pri != FEATURE_TRUE) post: result.status == RMI_ERROR_INPUT
dpt	pre: (params.flags.ats == RMI_FEATURE_TRUE && DptL0().state != DPT_L0_VALID) post: result.status == RMI_ERROR_DPT

B4.5.29.2.1 Failure condition ordering

The RMI_PSMMU_ACTIVATE command does not have any failure condition orderings.

B4.5.29.3 Success conditions

ID	Condition
state	post: psmmu.state == PSMMU_ACTIVE
gerr_addr	pre: params.flags.irq_cfg == RMI_IRQ_MSI post: psmmu.msi_config.gerr_addr == params.gerr_addr
gerr_data	pre: params.flags.irq_cfg == RMI_IRQ_MSI post: psmmu.msi_config.gerr_data == params.gerr_data
eventq_addr	pre: params.flags.irq_cfg == RMI_IRQ_MSI post: psmmu.msi_config.eventq_addr == params.eventq_addr
eventq_data	pre: params.flags.irq_cfg == RMI_IRQ_MSI post: psmmu.msi_config.eventq_data == params.eventq_data
priq_addr	pre: params.flags.irq_cfg == RMI_IRQ_MSI post: psmmu.msi_config.priq_addr == params.priq_addr
priq_data	pre: params.flags.irq_cfg == RMI_IRQ_MSI post: psmmu.msi_config.priq_data == params.priq_data

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.29.4 Footprint

The RMI_PSMMU_ACTIVATE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.30 RMI_PSMMU_DEACTIVATE command

Deactivate a PSMMU.

The RMI_PSMMU_DEACTIVATE command may initiate a Stateful RMI Operation.

The RMI_PSMMU_DEACTIVATE command may initiate a memory-transferring RMI Operation.

See also:

- A9.7 *Physical SMMU*
- B4.5.29 *RMI_PSMMU_ACTIVATE command*

B4.5.30.1 Interface

B4.5.30.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001D8`
psmmu_ptr	X1	63:0	Address	PA of PSMMU

B4.5.30.1.2 Context

The RMI_PSMMU_DEACTIVATE command operates on the following context.

Name	Type	Value	Before	Description
psmmu	RmmPsmmu	`PsmmuAt(psmmu_ptr)`	false	PSMMU

B4.5.30.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.30.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
psmmu_valid	`pre: !PsmmuAddrIsValid(psmmu_ptr)` `post: result.status == RMI_ERROR_INPUT`
psmmu_state	`pre: psmmu.state != PSMMU_ACTIVE` `post: result.status == RMI_ERROR_INPUT`
psmmu_live	`pre: PsmmuL1StIsLive(psmmu)` `post: result.status == RMI_ERROR_INPUT`

B4.5.30.2.1 Failure condition ordering

The RMI_PSMMU_DEACTIVATE command does not have any failure condition orderings.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.30.3 Success conditions

ID	Condition
state	`post: psmmu.state == PSMMU_INACTIVE`

B4.5.30.4 Footprint

The RMI_PSMMU_DEACTIVATE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.31 RMI_PSMMU_EVENT_CONSUME command

Consume a PSMMU event.

See also:

- A9.8.6 *Page Request Interface events*

B4.5.31.1 Interface

B4.5.31.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F0`
psmmu_ptr	X1	63:0	Address	PA of PSMMU
irq	X2	1:0	RmiPsmmuIrq	SMMU IRQ

The following unused bits of RMI_PSMMU_EVENT_CONSUME input values SBZ: X2[63:2].

B4.5.31.1.2 Context

The RMI_PSMMU_EVENT_CONSUME command operates on the following context.

Name	Type	Value	Before	Description
psmmu	RmmPsmmu	`PsmmuAt(psmmu_ptr)`	false	PSMMU

B4.5.31.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.31.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
psmmu_valid	`pre: !PsmmuAddrIsValid(psmmu_ptr)` `post: result.status == RMI_ERROR_INPUT`
psmmu_state	`pre: psmmu.state != PSMMU_ACTIVE` `post: result.status == RMI_ERROR_INPUT`

B4.5.31.2.1 Failure condition ordering

The RMI_PSMMU_EVENT_CONSUME command does not have any failure condition orderings.

B4.5.31.3 Success conditions

The RMI_PSMMU_EVENT_CONSUME command does not have any success conditions.

B4.5.31.4 Footprint

The RMI_PSMMU_EVENT_CONSUME command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.32 RMI_PSMMU_INFO command

Returns PSMMU info.

B4.5.32.1 Interface

B4.5.32.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400020E`
psmmu_ptr	X1	63:0	Address	PA of the PSMMU
info_ptr	X2	63:0	Address	PA of the PSMMU info structure

B4.5.32.1.2 Context

The RMI_PSMMU_INFO command operates on the following context.

Name	Type	Value	Before	Description
psmmu	RmmPsmmu	`PsmmuAt(psmmu_ptr)`	false	PSMMU
info	RmiPsmmuInfo	`PsmmuInfoAt(info_ptr)`	false	PSMMU info

B4.5.32.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

DRAFT

B4.5.32.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
psmmu_valid	`pre: !PsmmuAddrIsValid(psmmu_ptr)` `post: result.status == RMI_ERROR_INPUT`
info_align	`pre: !AddrIsAligned(info_ptr, 0x100)` `post: result.status == RMI_ERROR_INPUT`
info_pas	`pre: !NonSecureAccessPermitted(info_ptr)` `post: result.status == RMI_ERROR_INPUT`

B4.5.32.2.1 Failure condition ordering

The RMI_PSMMU_INFO command does not have any failure condition orderings.

B4.5.32.3 Success conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
irq_cfg	`post: Equal(info.flags.irq_cfg, psmmu.irq_cfg)`
cmdq_sync_irq	`post: Equal(info.flags.cmdq_sync_irq_wired, psmmu.cmdq_sync_irq_wired)`
gerror_intr_num	`post: info.gerror_intr_num == psmmu.gerror_intr_num`
eventq_intr_num	`post: info.eventq_intr_num == psmmu.eventq_intr_num`
priq_intr_num	`post: info.priq_intr_num == psmmu.priq_intr_num`
cmdq_sync_intr_num	`post: info.cmdq_sync_intr_num == psmmu.cmdq_sync_intr_num`

B4.5.32.4 Footprint

The RMI_PSMMU_INFO command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.33 RMI_PSMMU_IRQ_NOTIFY command

Notify RMM of an SMMU interrupt.

See also:

- A9.7.6 *PSMMU interrupts*
- A9.8.6 *Page Request Interface events*

B4.5.33.1 Interface

B4.5.33.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400016F`
psmmu_ptr	X1	63:0	Address	PA of PSMMU
irqs	X2	63:0	RmiPsmmuIrqSet	Set of pending PSMMU IRQs

B4.5.33.1.2 Context

The RMI_PSMMU_IRQ_NOTIFY command operates on the following context.

Name	Type	Value	Before	Description
psmmu	RmmPsmmu	`PsmmuAt(psmmu_ptr)`	false	PSMMU

B4.5.33.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
flags	X1	63:0	RmiPsmmuIrqResult	Result of triaging the IRQ
event_num	X2	63:0	UInt64	SMMU event number
sid	X3	63:0	Bits64	PSMMU Stream ID This is valid if flags.vsmmu == RMI_TRUE.
fetch_addr	X4	63:0	Address	Physical fetch address, as specified by SMMU
input_addr	X5	63:0	Address	Physical input address, as specified by SMMU
syndrome	X6	63:0	Bits64	RnW[0], S2[1] and Class[3:2] attributes, as specified by SMMU

B4.5.33.2 Failure conditions

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
feat	pre: Rmm().static.feat_da != FEATURE_TRUE post: result.status == RMI_ERROR_NOT_SUPPORTED
psmmu_valid	pre: !PsmmuAddrIsValid(psmmu_ptr) post: result.status == RMI_ERROR_INPUT
psmmu_state	pre: psmmu.state != PSMMU_ACTIVE post: result.status == RMI_ERROR_INPUT

B4.5.33.2.1 Failure condition ordering

The RMI_PSMMU_IRQ_NOTIFY command does not have any failure condition orderings.

B4.5.33.3 Success conditions

The RMI_PSMMU_IRQ_NOTIFY command does not have any success conditions.

B4.5.33.4 Footprint

The RMI_PSMMU_IRQ_NOTIFY command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.34 RMI_PSMMU_ST_L2_CREATE command

Create a PSMMU Level 2 Stream Table.

The RMI_PSMMU_ST_L2_CREATE command may initiate a Stateful RMI Operation.

The RMI_PSMMU_ST_L2_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- A9.7.4 *PSMMU Stream Tables*

B4.5.34.1 Interface

B4.5.34.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001DB`
psmmu_ptr	X1	63:0	Address	PA of PSMMU
sid	X2	63:0	Bits64	Base of StreamID range described by the Level 2 Stream Table

B4.5.34.1.2 Context

The RMI_PSMMU_ST_L2_CREATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
psmmu	RmmPsmmu	`PsmmuAt(psmmu_ptr)`	false	PSMMU
walk	RmmPsmmuStWalkResul	`PsmmuStWalk(psmmu, sid)`	false	Result of PSMMU Stream Table walk

DRAFT

B4.5.34.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.34.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
psmmu_valid	`pre: !PsmmuAddrIsValid(psmmu_ptr)` `post: result.status == RMI_ERROR_INPUT`
psmmu_state	`pre: psmmu.state != PSMMU_ACTIVE` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
sid_bound	`pre: UInt(sid) >= 2^psmmu.sid_size` `post: result.status == RMI_ERROR_INPUT`

st_entry	`pre: (walk.level != 1`
	`|| walk.ste.state == PSMMU_ST_ENTRY_TABLE)`
	`post: result.status == RMI_ERROR_INPUT`

sid_align	`pre: sid identifies the first entry in an L2ST.` `post: result.status == RMI_ERROR_INPUT`

B4.5.34.2.1 Failure condition ordering

The RMI_PSMMU_ST_L2_CREATE command does not have any failure condition orderings.

B4.5.34.3 Success conditions

ID	Condition
result	`post: result.status == RMI_SUCCESS`
state	`post: walk.ste.state == PSMMU_ST_ENTRY_TABLE`

B4.5.34.4 Footprint

ID	Value
ste_state	`walk.ste.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.35 RMI_PSMMU_ST_L2_DESTROY command

Destroy a PSMMU Level 2 Stream Table.

The RMI_PSMMU_ST_L2_DESTROY command may initiate a Stateful RMI Operation.

The RMI_PSMMU_ST_L2_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- A9.7.4 *PSMMU Stream Tables*

B4.5.35.1 Interface

B4.5.35.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001DC`
psmmu_ptr	X1	63:0	Address	PA of PSMMU
sid	X2	63:0	Bits64	Base of StreamID range described by the Level 2 Stream Table

B4.5.35.1.2 Context

The RMI_PSMMU_ST_L2_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
psmmu	RmmPsmmu	`PsmmuAt(psmmu_ptr)`	false	PSMMU
walk	RmmPsmmuStWalkResul	`PsmmuStWalk(psmmu, sid)`	false	Result of PSMMU Stream Table walk

DRAFT

B4.5.35.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.35.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
psmmu_valid	`pre: !PsmmuAddrIsValid(psmmu_ptr)` `post: result.status == RMI_ERROR_INPUT`
sid_bound	`pre: UInt(sid) >= 2^psmmu.sid_size` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
sid_align	pre: sid identifies the first entry in an L2ST. post: result.status == RMI_ERROR_INPUT
psmmu_state	pre: psmmu.state != PSMMU_ACTIVE post: result.status == RMI_ERROR_INPUT
st_entry	pre: walk.ste.state == PSMMU_ST_ENTRY_INVALID post: result.status == RMI_ERROR_INPUT
l2st_live	pre: PsmmuL2StIsLive(psmmu, sid) post: result.status == RMI_ERROR_INPUT

B4.5.35.2.1 Failure condition ordering

The RMI_PSMMU_ST_L2_DESTROY command does not have any failure condition orderings.

B4.5.35.3 Success conditions

ID	Condition
result	post: result.status == RMI_SUCCESS
state	post: walk.ste.state == PSMMU_ST_ENTRY_INVALID

B4.5.35.4 Footprint

ID	Value
ste_state	walk.ste.state

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.36 RMI_REALM_ACTIVATE command

Activates a Realm.

See also:

- A2.2 *Realm*
- B4.5.39 *RMI_REALM_TERMINATE command*

B4.5.36.1 Interface

B4.5.36.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000157`
rd	X1	63:0	Address	PA of the RD

B4.5.36.1.2 Context

The RMI_REALM_ACTIVATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm

B4.5.36.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.36.2 Failure conditions

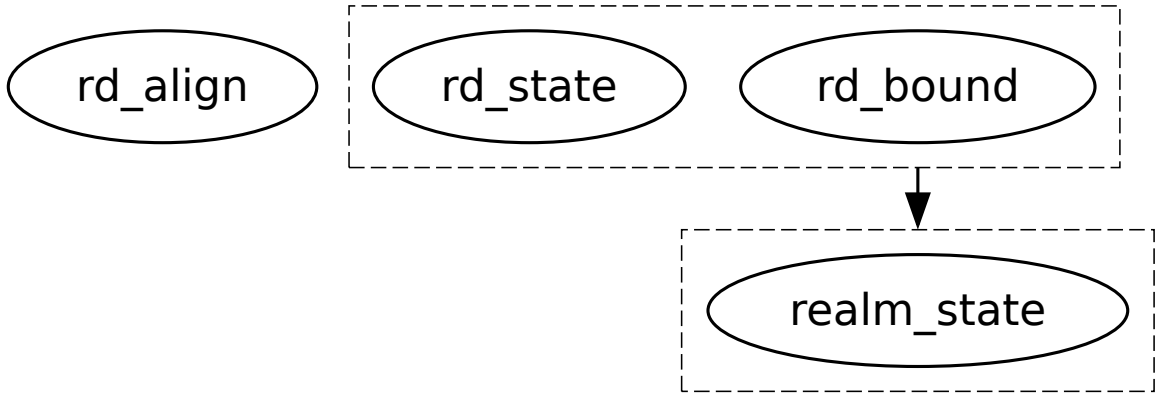
ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
realm_state	`pre: realm.state != REALM_NEW` `post: result.status == RMI_ERROR_REALM`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.36.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
```



B4.5.36.3 Success conditions

ID	Condition
realm_state	`post: realm.state == REALM_ACTIVE`

B4.5.36.4 Footprint

ID	Value
realm_state	`realm.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.37 RMI_REALM_CREATE command

Creates a Realm.

The RMI_REALM_CREATE command may initiate a Stateful RMI Operation.

The RMI_REALM_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- A2.2 *Realm*
- A2.2.6 *Realm parameters*
- B4.3.4 *Object creation and destruction*
- B4.5.38 *RMI_REALM_DESTROY command*
- D1.2.1 *Realm creation flow*

B4.5.37.1 Interface

B4.5.37.1.1 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

B4.5.37.1.2 Context

The RMI_REALM_CREATE command operates on the following context.

Name	Type	Value	Before	Description
rmm_pre	RmmGlobal	`Rmm()`	true	RMM global state
rmm	RmmGlobal	`Rmm()`	false	RMM global state
params	RmiRealmParams	`RmiRealmParamsAt(params_ptr)`	false	Realm parameters
realm	RmmRealm	`RealmAt(rd)`	false	Realm

B4.5.37.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.37.2 Failure conditions

ID	Condition
params_align	`pre: !AddrIsRmiGranuleAligned(params_ptr)` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
params_pas	pre: !NonSecureAccessPermitted(params_ptr) post: result.status == RMI_ERROR_INPUT
params_valid	pre: !RmiRealmParamsIsValid(params_ptr) post: result.status == RMI_ERROR_INPUT
params_supp	pre: !RealmParamsSupported(params) post: result.status == RMI_ERROR_INPUT
alias	pre: AddrInRange(rd, params.rtt_base, (params.rtt_num_start - 1) * rmm.dynamic.rmi_granule_size) post: result.status == RMI_ERROR_INPUT
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsDelegableConventionalFine(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_DELEGATED post: result.status == RMI_ERROR_INPUT
rtt_align	pre: !AddrIsAligned(params.rtt_base, params.rtt_num_start * rmm.dynamic.rmi_granule_size) post: result.status == RMI_ERROR_INPUT
rtt_num_level	pre: !RttConfigIsValid(params.s2sz, params.rtt_level_start, params.rtt_num_start) post: result.status == RMI_ERROR_INPUT
rtt_state	pre: !RttsStateEqual(params.rtt_base, params.rtt_num_start, GRAN_DELEGATED) post: result.status == RMI_ERROR_INPUT
ats_plane	pre: params.ats_plane > params.num_aux_planes post: result.status == RMI_ERROR_INPUT
vmid	pre: !VmidsAvailable(params.num_aux_planes + 1) post: result.status == RMI_ERROR_GLOBAL
mec_policy	pre: !MecidAvailable(params.flags0.mec_policy) post: result.status == RMI_ERROR_GLOBAL

B4.5.37.2.1 Failure condition ordering

The RMI_REALM_CREATE command does not have any failure condition orderings.

B4.5.37.3 Success conditions

ID	Condition
num_realms	post: rmm.dynamic.num_realms == rmm_pre.dynamic.num_realms + 1
rd_state	post: GranuleAt(rd).state == GRAN_RD
realm_state	post: realm.state == REALM_NEW
rec_index	post: realm.rec_index == 0
rtt_base	post: RealmRttBaseEqual(realm, params.rtt_base, params.aux_rtt_base)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rtt_state	`post: RttsStateEqual(realm.rtt_base[[0]], realm.rtt_num_start, GRAN_RTT)`
rtte_p_states	`post: RttsAllProtectedEntriesState(realm.rtt_base[[0]], realm.rtt_num_start, RTTE_VOID)`
rtte_up_states	`post: RttsAllUnprotectedEntriesState(realm.rtt_base[[0]], realm.rtt_num_start, RTTE_UNMAPPED_NS)`
rtte_ripas	`post: RttsAllProtectedEntriesRipas(realm.rtt_base[[0]], realm.rtt_num_start, RIPAS_EMPTY)`
lpa2	`post: Equal(realm.feat_lpa2, params.flags0.lpa2)`
ipa_width	`post: realm.ipa_width == params.s2sz`
hash_algo	`post: Equal(realm.hash_algo, params.hash_algo)`
rim	`post: realm.rim == Zeros{RMM_REALM_MEASUREMENT_WIDTH}()`
rem	`post: (realm.rem[[0]] == Zeros{RMM_REALM_MEASUREMENT_WIDTH}() && realm.rem[[1]] == Zeros{RMM_REALM_MEASUREMENT_WIDTH}() && realm.rem[[2]] == Zeros{RMM_REALM_MEASUREMENT_WIDTH}() && realm.rem[[3]] == Zeros{RMM_REALM_MEASUREMENT_WIDTH}())`
rtt_level	`post: realm.rtt_level_start == params.rtt_level_start`
rtt_num	`post: realm.rtt_num_start == params.rtt_num_start`
rpv	`post: realm.rpv == params.rpv`
da	`post: Equal(realm.feat_da, params.flags0.da)`
ats	`post: Equal(realm.feat_ats, params.flags1.ats)`
ats_plane	`post: realm.ats_plane == params.ats_plane`
rtt_tree_per_plane	`post: Equal(realm.rtt_tree_per_plane, params.flags1.rtt_tree_per_plane)`
num_aux_planes	`post: realm.num_aux_planes == params.num_aux_planes`
rtt_s2ap_encoding	`post: Equal(realm.rtt_s2ap_encoding, params.flags1.rtt_s2ap_encoding)`
lfa_policy	`post: Equal(realm.lfa_policy, params.flags0.lfa_policy)`
mec_policy	`post: Equal(realm.mec_policy, params.flags0.mec_policy)`
num_recs	`post: realm.num_recs == 0`
num_vdevs	`post: realm.num_vdevs == 0`
num_vsmmus	`post: realm.num_vsmmus == 0`

B4.5.37.4 RMI_REALM_CREATE initialization of RIM

On successful execution of RMI_REALM_CREATE, the initial RIM value of the target Realm is zero.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- A7.1.1 *Realm Initial Measurement*

B4.5.37.5 Footprint

ID	Value
rd_state	`GranuleAt(rd).state`
rtt_state	`RttsGranuleState( realm.rtt_base[[0]], realm.rtt_num_start)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.38 RMI_REALM_DESTROY command

Destroys a Realm.

The RMI_REALM_DESTROY command may initiate a Stateful RMI Operation.

The RMI_REALM_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- A2.2 *Realm*
- B4.3.4 *Object creation and destruction*
- B4.5.37 *RMI_REALM_CREATE command*
- D1.2.5 *Realm destruction flow*

B4.5.38.1 Interface

B4.5.38.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000159`
rd	X1	63:0	Address	PA of the RD

B4.5.38.1.2 Context

The RMI_REALM_DESTROY command operates on the following context.

Name	Type	Value	Before	Description
rmm_pre	RmmGlobal	`Rmm()`	true	RMM global state
rmm	RmmGlobal	`Rmm()`	false	RMM global state
realm_pre	RmmRealm	`RealmAt(rd)`	true	Realm
realm	RmmRealm	`RealmAt(rd)`	false	Realm

B4.5.38.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.38.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_tracking	`pre: !PaIsTrackedFine(rd)` `post: result.status == RMI_ERROR_INPUT`

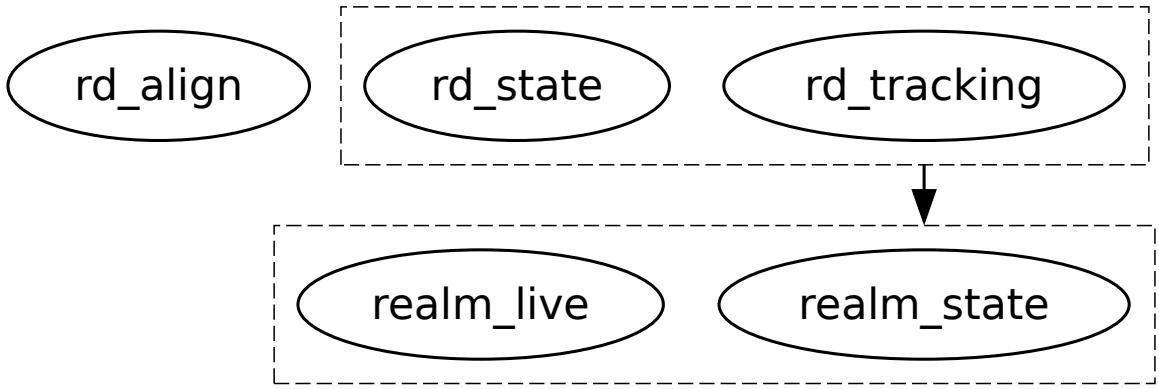
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm.state != REALM_ZOMBIE post: result.status == RMI_ERROR_REALM
realm_live	pre: RealmIsLive(rd) post: result.status == RMI_ERROR_REALM

B4.5.38.2.1 Failure condition ordering

```
[rd_tracking, rd_state] < [realm_state, realm_live]
```



B4.5.38.3 Success conditions

ID	Condition
result	post: result.status == RMI_SUCCESS
num_realms	post: rmm.dynamic.num_realms == rmm_pre.dynamic.num_realms - 1
rtt_state	post: RttsStateEqual(realm_pre.rtt_base[[0]], realm_pre.rtt_num_start, GRAN_DELEGATED)
rd_state	post: GranuleAt(rd).state == GRAN_DELEGATED

B4.5.38.4 Footprint

ID	Value
rd_state	GranuleAt(rd).state
rtt_state	RttsGranuleState(realm_pre.rtt_base[[0]], realm_pre.rtt_num_start)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.39 RMI_REALM_TERMINATE command

Terminates a Realm.

The RMI_REALM_TERMINATE command may initiate a Stateful RMI Operation.

See also:

- A2.2 *Realm*
- B4.5.36 *RMI_REALM_ACTIVATE command*
- D1.2.5 *Realm destruction flow*

B4.5.39.1 Interface

B4.5.39.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000201`
rd	X1	63:0	Address	PA of the RD

B4.5.39.1.2 Context

The RMI_REALM_TERMINATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm

B4.5.39.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.39.2 Failure conditions

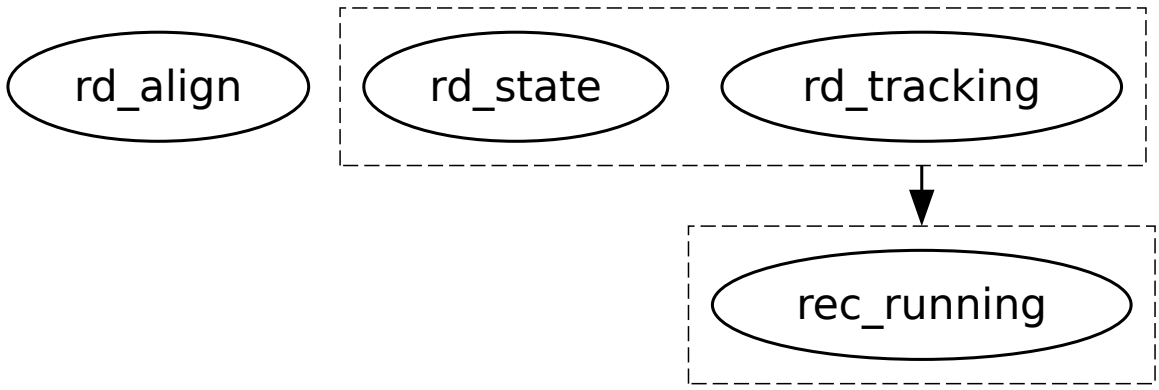
ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_tracking	`pre: !PaIsTrackedFine(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
rec_running	`pre: AnyRecRunning(realm)` `post: result.status == RMI_ERROR_REALM`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.39.2.1 Failure condition ordering

```
[rd_tracking, rd_state] < [rec_running]
```



B4.5.39.3 Success conditions

ID	Condition
realm_state	`post: realm.state == REALM_ZOMBIE`
result	`post: result.status == RMI_SUCCESS`

B4.5.39.4 Footprint

ID	Value
realm_state	`realm.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.40 RMI_REC_CREATE command

Creates a REC.

The RMI_REC_CREATE command may initiate a Stateful RMI Operation.

The RMI_REC_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- A2.4 *Realm Execution Context*
- A2.4.3 *REC index and MPIDR value*
- B4.3.4 *Object creation and destruction*
- B4.5.41 *RMI_REC_DESTROY command*
- D1.2.4 *REC creation flow*

B4.5.40.1 Interface

B4.5.40.1.1 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

DRAFT

B4.5.40.1.2 Context

The RMI_REC_CREATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
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

B4.5.40.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.40.2 Failure conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
params_align	pre: !AddrIsRmiGranuleAligned(params_ptr) post: result.status == RMI_ERROR_INPUT
params_pas	pre: !NonSecureAccessPermitted(params_ptr) post: result.status == RMI_ERROR_INPUT
rec_align	pre: !AddrIsRmiGranuleAligned(rec_ptr) post: result.status == RMI_ERROR_INPUT
rec_bound	pre: !PaIsDelegableConventionalFine(rec_ptr) post: result.status == RMI_ERROR_INPUT
rec_state	pre: GranuleAt(rec_ptr).state != GRAN_DELEGATED post: result.status == RMI_ERROR_INPUT
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm_pre.state != REALM_NEW post: result.status == RMI_ERROR_REALM
num_recs	pre: realm_pre.num_recs == (2 ^ rmm.static.max_recs_order) - 1 post: result.status == RMI_ERROR_REALM
mpidr_index	pre: RecIndex(params.mpidr) != realm_pre.rec_index post: result.status == RMI_ERROR_INPUT

B4.5.40.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state, num_recs]
```

B4.5.40.3 Success conditions

ID	Condition
rec_index	post: realm.rec_index == realm_pre.rec_index + 1
rec_gran_state	post: GranuleAt(rec_ptr).state == GRAN_REC
rec_owner	post: rec.owner == rd
rec_attest	post: rec.attest_state == NO_ATTEST_IN_PROGRESS
rec_mpidr	post: MpidrEqual(rec.mpidr, params.mpidr)
rec_state	post: rec.state == REC_READY
runnable	pre: params.flags.runnable == RMI_RUNNABLE post: rec.flags.runnable == RUNNABLE

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
not_runnable	`pre: params.flags.runnable == RMI_NOT_RUNNABLE` `post: rec.flags.runnable == NOT_RUNNABLE`

rec_gprs	`post: (rec.gprs[[0]] == params.gprs[[0]`
	`&& rec.gprs[[1]] == params.gprs[[1]]`
	`&& rec.gprs[[2]] == params.gprs[[2]]`
	`&& rec.gprs[[3]] == params.gprs[[3]]`
	`&& rec.gprs[[4]] == params.gprs[[4]]`
	`&& rec.gprs[[5]] == params.gprs[[5]]`
	`&& rec.gprs[[6]] == params.gprs[[6]]`
	`&& rec.gprs[[7]] == params.gprs[[7]]`
	`&& rec.gprs[[8]] == Zeros{64}()`
	`&& rec.gprs[[9]] == Zeros{64}()`
	`&& rec.gprs[[10]] == Zeros{64}()`
	`&& rec.gprs[[11]] == Zeros{64}()`
	`&& rec.gprs[[12]] == Zeros{64}()`
	`&& rec.gprs[[13]] == Zeros{64}()`
	`&& rec.gprs[[14]] == Zeros{64}()`
	`&& rec.gprs[[15]] == Zeros{64}()`
	`&& rec.gprs[[16]] == Zeros{64}()`
	`&& rec.gprs[[17]] == Zeros{64}()`
	`&& rec.gprs[[18]] == Zeros{64}()`
	`&& rec.gprs[[19]] == Zeros{64}()`
	`&& rec.gprs[[20]] == Zeros{64}()`
	`&& rec.gprs[[21]] == Zeros{64}()`
	`&& rec.gprs[[22]] == Zeros{64}()`
	`&& rec.gprs[[23]] == Zeros{64}()`
	`&& rec.gprs[[24]] == Zeros{64}()`
	`&& rec.gprs[[25]] == Zeros{64}()`
	`&& rec.gprs[[26]] == Zeros{64}()`
	`&& rec.gprs[[27]] == Zeros{64}()`
	`&& rec.gprs[[28]] == Zeros{64}()`
	`&& rec.gprs[[29]] == Zeros{64}()`
	`&& rec.gprs[[30]] == Zeros{64}()`
	`&& rec.gprs[[31]] == Zeros{64}())`

rec_pc	`post: rec.pc == params.pc`
rim	`pre: params.flags.runnable == RMI_RUNNABLE` `post: realm.rim == RimExtendRec(realm_pre, params)`
ripas_addr	`post: rec.ripas_addr == Zeros{ADDRESS_WIDTH}()`
ripas_top	`post: rec.ripas_top == Zeros{ADDRESS_WIDTH}()`
pending	`post: rec.pending == REC_PENDING_NONE`
num_recs	`post: realm.num_recs == realm_pre.num_recs + 1`
gic_owner	`post: rec.gic_owner == 0`

B4.5.40.4 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:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- 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*
- B3.107 *RimExtendRec function*
- B4.6.69 *RmiRecParams type*
- C2.26 *RmmMeasurementDescriptorRec type*

B4.5.40.5 Footprint

ID	Value
rec_index	`realm.rec_index`
rec_state	`GranuleAt(rec).state`
rim	`realm.rim`
num_recs	`realm.num_recs`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.41 RMI_REC_DESTROY command

Destroys a REC.

The RMI_REC_DESTROY command may initiate a Stateful RMI Operation.

The RMI_REC_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- A2.4 *Realm Execution Context*
- B4.3.4 *Object creation and destruction*
- B4.5.40 *RMI_REC_CREATE command*
- D1.2.5 *Realm destruction flow*

B4.5.41.1 Interface

B4.5.41.1.1 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

B4.5.41.1.2 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

B4.5.41.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.41.2 Failure conditions

ID	Condition
rec_align	`pre: !AddrIsRmiGranuleAligned(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_tracking	`pre: !PaIsTrackedFine(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`

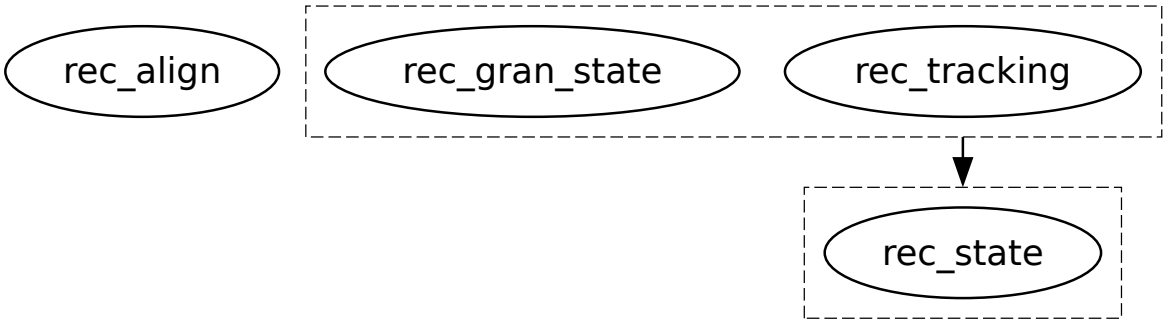
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rec_gran_state	pre: GranuleAt(rec_ptr).state != GRAN_REC post: result.status == RMI_ERROR_INPUT
rec_state	pre: rec.state == REC_RUNNING post: result.status == RMI_ERROR_REC

B4.5.41.2.1 Failure condition ordering

```
[rec_tracking, rec_gran_state] < [rec_state]
```



B4.5.41.3 Success conditions

ID	Condition
rec_gran_state	post: GranuleAt(rec_ptr).state == GRAN_DELEGATED
num_recs	post: realm.num_recs == realm_pre.num_recs - 1

B4.5.41.4 Footprint

ID	Value
rec_state	GranuleAt(rec_ptr).state
num_recs	realm.num_recs

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.42 RMI_REC_ENTER command

Enter a REC.

See also:

- A2.4 *Realm Execution Context*
- Chapter A4 *Realm exception model*
- D1.3.1 *Realm entry and exit flow*

B4.5.42.1 Interface

B4.5.42.1.1 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

B4.5.42.1.2 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

DRAFT

B4.5.42.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.42.2 Failure conditions

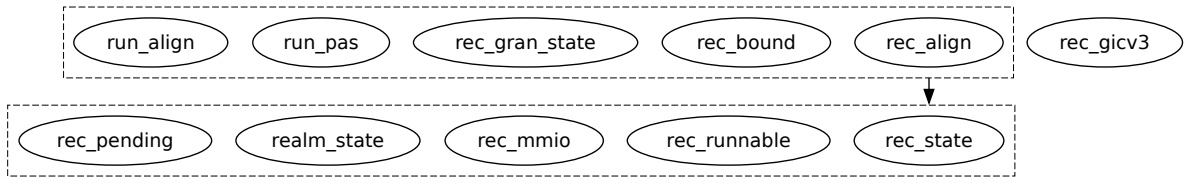
ID	Condition
run_align	`pre: !AddrIsRmiGranuleAligned(run_ptr)` `post: result.status == RMI_ERROR_INPUT`
run_pas	`pre: !NonSecureAccessPermitted(run_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_align	`pre: !AddrIsRmiGranuleAligned(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_bound	`pre: !PaIsTracked(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rec_gran_state	pre: GranuleAt(rec_ptr).state != GRAN_REC post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm.state != REALM_ACTIVE post: result.status == RMI_ERROR_REALM
rec_state	pre: rec.state == REC_RUNNING post: result.status == RMI_ERROR_REC
rec_runnable	pre: rec.flags.runnable == NOT_RUNNABLE post: result.status == RMI_ERROR_REC
rec_mmio	pre: (run.enter.flags.emul_mmio == RMI_EMULATED_MMIO && rec.emulatable_abort != EMULATABLE_ABORT) post: result.status == RMI_ERROR_REC
rec_gicv3	pre: !Gicv3ConfigIsValid() post: result.status == RMI_ERROR_REC
rec_pending	pre: rec.pending == REC_PENDING_PSCI post: result.status == RMI_ERROR_REC

B4.5.42.2.1 Failure condition ordering

```
[rec_align, rec_bound, rec_gran_state, run_pas, run_align] <
     [rec_state, rec_runnable, rec_mmio, realm_state, rec_pending]
```



B4.5.42.3 Success conditions

ID	Condition
rec_exit	post: run.exit contains Realm exit syndrome information.
rec_emul_abt	post: rec.emulatable_abort is updated.

B4.5.42.4 Footprint

ID	Value
emul_abt	rec.emulatable_abort

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.43 RMI_RMM_ACTIVATE command

Activate the RMM.

The RMI_RMM_ACTIVATE command may initiate a Stateful RMI Operation.

The RMI_RMM_ACTIVATE command may initiate a memory-transferring RMI Operation.

See also:

- A2.1 *RMM*

B4.5.43.1 Interface

B4.5.43.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000202`

B4.5.43.1.2 Context

The RMI_RMM_ACTIVATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state

B4.5.43.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.43.2 Failure conditions

ID	Condition
state	`pre: rmm.dynamic.state != RMM_STATE_INIT` `post: result.status == RMI_ERROR_GLOBAL`

B4.5.43.3 Success conditions

ID	Condition
result	`post: result.status == RMI_SUCCESS`
state	`post: rmm.dynamic.state == RMM_STATE_ACTIVE`

B4.5.43.4 Footprint

The RMI_RMM_ACTIVATE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.44 RMI_RMM_CONFIG_GET command

Get the system configuration.

See also:

- A2.1 *RMM*

B4.5.44.1 Interface

B4.5.44.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001EC`
cfg_ptr	X1	63:0	Address	PA of configuration structure

B4.5.44.1.2 Context

The RMI_RMM_CONFIG_GET command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
cfg	RmiRmmConfig	`RmiRmmConfigAt(cfg_ptr)`	false	Configuration

B4.5.44.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.44.2 Failure conditions

ID	Condition
rmm_state	`pre: rmm.dynamic.state != RMM_STATE_ACTIVE` `post: result.status == RMI_ERROR_GLOBAL`
cfg_align	`pre: !AddrIsRmiGranuleAligned(cfg_ptr)` `post: result.status == RMI_ERROR_INPUT`
cfg_pas	`pre: !NonSecureAccessPermitted(cfg_ptr)` `post: result.status == RMI_ERROR_INPUT`

B4.5.44.2.1 Failure condition ordering

The RMI_RMM_CONFIG_GET command does not have any failure condition orderings.

B4.5.44.3 Success conditions

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rmi_granule_size	`post: cfg.rmi_granule_size == GranuleSizeToRmi(rmm.dynamic.rmi_granule_size)`
tracking_region_size	`post: cfg.tracking_region_size == TrackingRegionSizeToRmi(rmm.dynamic.rmi_granule_size, rmm.dynamic.tracking_region_size)`

B4.5.44.4 Footprint

The RMI_RMM_CONFIG_GET command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.45 RMI_RMM_CONFIG_SET command

Set the system configuration.

See also:

- A2.1 *RMM*

B4.5.45.1 Interface

B4.5.45.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400016E`
cfg_ptr	X1	63:0	Address	PA of configuration structure

B4.5.45.1.2 Context

The RMI_RMM_CONFIG_SET command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
cfg	RmiRmmConfig	`RmiRmmConfigAt(cfg_ptr)`	false	Configuration

B4.5.45.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.45.2 Failure conditions

ID	Condition
rmm_state	`pre: rmm.dynamic.state != RMM_STATE_INIT` `post: result.status == RMI_ERROR_GLOBAL`
cfg_align	`pre: !AddrIsRmiGranuleAligned(cfg_ptr)` `post: result.status == RMI_ERROR_INPUT`
cfg_pas	`pre: !NonSecureAccessPermitted(cfg_ptr)` `post: result.status == RMI_ERROR_INPUT`
cfg_supp	`pre: !RmmConfigIsSupported(cfg)` `post: result.status == RMI_ERROR_INPUT`
tracked	`pre: rmm.dynamic.num_tracked != 0` `post: result.status == RMI_ERROR_INPUT`

B4.5.45.2.1 Failure condition ordering

The RMI_RMM_CONFIG_SET command does not have any failure condition orderings.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.45.3 Success conditions

ID	Condition
rmi_granule_size	`post: rmm.dynamic.rmi_granule_size == GranuleSizeFromRmi(cfg.rmi_granule_size)`
tracking_region_size	`post: rmm.dynamic.tracking_region_size == TrackingRegionSizeFromRmi(cfg.rmi_granule_size, cfg.tracking_region_size)`

B4.5.45.4 Footprint

ID	Value
rmi_granule_size	`rmm.dynamic.rmi_granule_size`
tracking_region_size	`rmm.dynamic.tracking_region_size`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.46 RMI_RMM_STATE_GET command

Get the state of the RMM.

B4.5.46.1 Interface

B4.5.46.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001EE`

B4.5.46.1.2 Context

The RMI_RMM_STATE_GET command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state

B4.5.46.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
state	X1	7:0	RmiRmmState	RMM state

The following unused bits of RMI_RMM_STATE_GET output values MBZ: X1[63:8].

B4.5.46.2 Failure conditions

The RMI_RMM_STATE_GET command does not have any failure conditions.

B4.5.46.3 Success conditions

ID	Condition
result	`post: result.status == RMI_SUCCESS`
state	`post: Equal(state, rmm.dynamic.state)`

B4.5.46.4 Footprint

The RMI_RMM_STATE_GET command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.47 RMI_RTT_ARCH_DEV_MAP command

Create mappings to an architectural device within a target Protected IPA range.

See also:

- A5.3.9 *Create mappings from Protected IPA space to an architectural device*
- A9.8 *Virtual SMMU*
- B4.5.48 *RMI_RTT_ARCH_DEV_UNMAP command*

B4.5.47.1 Interface

B4.5.47.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F9`
rd	X1	63:0	Address	PA of the RD for the target Realm
dev_ptr	X2	63:0	Address	PA of the device
base	X3	63:0	Address	Base of the target IPA range
top	X4	63:0	Address	Top of the target IPA range

B4.5.47.1.2 Context

The RMI_RTT_ARCH_DEV_MAP command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
realm	RmmRealm	`RealmAt(rd)`	false	Realm
vsmmu	RmmVsmmu	`VsmmuAt(dev_ptr)`	false	VSMMU
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes

walk	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of RTT walk from base of the target IPA range

B4.5.47.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

I_{JJQSG} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.47.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_vsmmu != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
dev_align	`pre: !AddrIsRmiGranuleAligned(dev_ptr)` `post: result.status == RMI_ERROR_INPUT`
dev_bound	`pre: !PaIsTracked(dev_ptr)` `post: result.status == RMI_ERROR_INPUT`
dev_state	`pre: GranuleAt(dev_ptr).state != GRAN_VSMMU` `post: result.status == RMI_ERROR_INPUT`
dev_realm	`pre: vsmmu.realm != rd` `post: result.status == RMI_ERROR_INPUT`
base_align	`pre: !AddrIsRmiGranuleAligned(base)` `post: result.status == RMI_ERROR_INPUT`
top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: !AddrRangeIsProtected(base, top, realm)` `post: result.status == RMI_ERROR_INPUT`

rtte_state	`pre: walk.rtte.state != RTTE_VOID`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

rtte_size	`pre: (walk.rtte.state == RTTE_VOID`
	`&& RttLevelSize(walk.level) > size)`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

rtte_ripas	`pre: walk.rtte.ripas != RIPAS_EMPTY`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

B4.5.47.2.1 Failure condition ordering

The RMI_RTT_ARCH_DEV_MAP command does not have any failure condition orderings.

B4.5.47.3 Success conditions

ID	Condition

state	`post: RttTreeRangeAllState(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base, out_top, RTTE_ARCH_DEV)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
addr	`post: RttTreeRangeAllOaddr( realm, RMM_RTT_TREE_PRIMARY, base, top, dev_ptr)`
result	`post: result.status == RMI_SUCCESS`

B4.5.47.4 Footprint

The RMI_RTT_ARCH_DEV_MAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.48 RMI_RTT_ARCH_DEV_UNMAP command

Removes mappings to an architectural device within a target Protected IPA range.

See also:

- A5.3.10 *Remove mappings from Protected IPA space to an architectural device*
- A9.8 *Virtual SMMU*
- B4.5.47 *RMI_RTT_ARCH_DEV_MAP command*

B4.5.48.1 Interface

B4.5.48.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001FA`
rd	X1	63:0	Address	PA of the RD for the target Realm
dev_ptr	X2	63:0	Address	PA of the device
base	X3	63:0	Address	Base of the target IPA range
top	X4	63:0	Address	Top of the target IPA range

B4.5.48.1.2 Context

The RMI_RTT_ARCH_DEV_UNMAP command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
realm_pre	RmmRealm	`RealmAt(rd)`	true	Realm
realm	RmmRealm	`RealmAt(rd)`	false	Realm
vsmmu	RmmVsmmu	`VsmmuAt(dev_ptr)`	false	VSMMU
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes

walk	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of RTT walk from base of the target IPA range

B4.5.48.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

I_{BYNQV} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.48.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
dev_align	`pre: !AddrIsRmiGranuleAligned(dev_ptr)` `post: result.status == RMI_ERROR_INPUT`
dev_bound	`pre: !PaIsTracked(dev_ptr)` `post: result.status == RMI_ERROR_INPUT`
dev_state	`pre: GranuleAt(dev_ptr).state != GRAN_VSMMU` `post: result.status == RMI_ERROR_INPUT`
dev_realm	`pre: vsmmu.realm != rd` `post: result.status == RMI_ERROR_INPUT`
base_align	`pre: !AddrIsRmiGranuleAligned(base)` `post: result.status == RMI_ERROR_INPUT`
top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: !AddrRangeIsProtected(base, top, realm)` `post: result.status == RMI_ERROR_INPUT`

rtte_state	`pre: walk.rtte.state != RTTE_ARCH_DEV`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

rtte_addr	`pre: (walk.rtte.state == RTTE_NARCH_DEV`
	`&& walk.rtte.addr != dev_ptr)`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

rtte_size	`pre: (walk.rtte.state == RTTE_NARCH_DEV`
	`&& RttLevelSize(walk.level) > size)`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

B4.5.48.2.1 Failure condition ordering

The RMI_RTT_ARCH_DEV_UNMAP command does not have any failure condition orderings.

B4.5.48.3 Success conditions

ID	Condition

state	`post: RttTreeRangeAllState(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base, out_top, RTTE_VOID)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
ripas	`post: RealmIpaRangeAllRipasIf(realm_pre, realm, base, out_top, RIPAS_DEV, RIPAS_DESTROYED)`
dev_state	`post: vsmmu.state == VSMMU_INACTIVE`
result	`post: result.status == RMI_SUCCESS`

B4.5.48.4 Footprint

The RMI_RTT_ARCH_DEV_UNMAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.49 RMI_RTT_AUX_CREATE command

Creates an auxiliary RTT.

The RMI_RTT_AUX_CREATE command may initiate a Stateful RMI Operation.

See also:

- A10.3.1 *Auxiliary RTT*
- B4.5.50 *RMI_RTT_AUX_DESTROY command*
- B4.5.51 *RMI_RTT_AUX_FOLD command*

B4.5.49.1 Interface

B4.5.49.1.1 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

DRAFT

B4.5.49.1.2 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

B4.5.49.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.49.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

level_bound	`pre: (!RttLevelIsValid(realm, level)`
	`|| RttLevelIsStarting(realm, level))`
	`post: result.status == RMI_ERROR_INPUT`

ipa_align	`pre: !AddrIsRttLevelAligned(ipa, level - 1)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: !AddrIsProtected(ipa, realm)` `post: result.status == RMI_ERROR_INPUT`

index_bound	`pre: (realm.rtt_tree_per_plane == FEATURE_FALSE`
	`|| index == RMM_RTT_TREE_PRIMARY`
	`|| index > realm.num_aux_planes)`
	`post: result.status == RMI_ERROR_INPUT`

rtt_align	`pre: !AddrIsRmiGranuleAligned(rtt)` `post: result.status == RMI_ERROR_INPUT`
rtt_bound	`pre: !PaIsDelegableConventionalFine(rtt)` `post: result.status == RMI_ERROR_INPUT`
rtt_state	`pre: GranuleAt(rtt).state != GRAN_DELEGATED` `post: result.status == RMI_ERROR_INPUT`

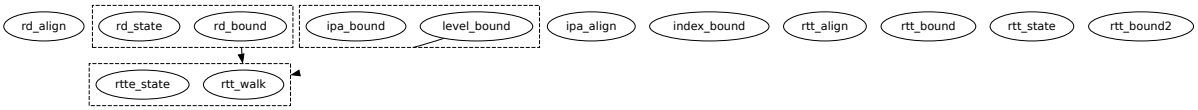
rtt_bound2	`pre: ((realm.feat_lpa2 == FEATURE_FALSE)`
	`&& (UInt(rtt) >= 2^48))`
	`post: result.status == RMI_ERROR_INPUT`

rtt_walk	`pre: walk.level < level - 1`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`&& result.data.level.level == walk.level)`

rtte_state	`pre: walk.rtte.state == RTTE_TABLE`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`&& result.data.level.level == walk.level)`

B4.5.49.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.5.49.3 Success conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rtt_state	`post: GranuleAt(rtt).state == GRAN_RTT`
rtte_addr	`post: walk.rtte.addr == rtt`
result	`post: result.status == RMI_SUCCESS`
rtte_state	`post: walk.rtte.state == RTTE_TABLE`
rtte_c_ripas	`pre: AddrIsProtected(ipa, realm)` `post: RttAllEntriesRipas(RttAt(rtt), unfold.ripas)`
rtte_c_state	`post: RttAllEntriesState(RttAt(rtt), unfold.state)`
rtte_c_addr	`pre: unfold.state != RTTE_VOID` `post: RttAllEntriesContiguous(RttAt(rtt), unfold.addr, level)`

B4.5.49.4 Footprint

ID	Value
rtt_state	`GranuleAt(rtt).state`
rtte	`RttEntry(walk.rtt_addr, entry_idx)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.50 RMI_RTT_AUX_DESTROY command

Destroys an auxiliary RTT.

The RMI_RTT_AUX_DESTROY command may initiate a Stateful RMI Operation.

See also:

- A10.3.1 *Auxiliary RTT*
- B4.5.49 *RMI_RTT_AUX_CREATE command*
- B4.5.51 *RMI_RTT_AUX_FOLD command*

B4.5.50.1 Interface

B4.5.50.1.1 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

B4.5.50.1.2 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

B4.5.50.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
rtt	X1	63:0	Address	PA of the RTT which was destroyed

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

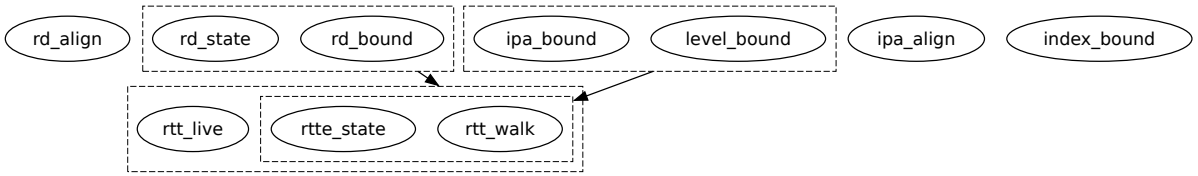
Name	Register	Bits	Type	Description
top	X2	63:0	Address	Top IPA of non-live RTT entries, from entry at which the RTT walk terminated

B4.5.50.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
level_bound	pre: (!RttLevelIsValid(realm, level) || RttLevelIsStarting(realm, level)) post: result.status == RMI_ERROR_INPUT
ipa_align	pre: !AddrIsRttLevelAligned(ipa, level - 1) post: result.status == RMI_ERROR_INPUT
ipa_bound	pre: !AddrIsProtected(ipa, realm) post: result.status == RMI_ERROR_INPUT
index_bound	pre: (realm.rtt_tree_per_plane == FEATURE_FALSE || index == RMM_RTT_TREE_PRIMARY || index > realm.num_aux_planes) post: result.status == RMI_ERROR_INPUT
rtt_walk	pre: walk.level < level - 1 post: (result.status == RMI_ERROR_RTT_AUX && result.data.level.level == walk.level && top == walk_top)
rtte_state	pre: walk.rtte.state != RTTE_TABLE post: (result.status == RMI_ERROR_RTT_AUX && result.data.level.level == walk.level && top == walk_top)
rtt_live	pre: RttIsLive(RttAt(walk.rtte.addr)) post: (result.status == RMI_ERROR_RTT_AUX && result.data.level.level == level && top == ipa)

B4.5.50.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state, rtt_live]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.50.3 Success conditions

ID	Condition
ripas	`post: walk.rtte.ripas == RIPAS_DESTROYED`
top	`post: top == walk_top`
rtt	`post: rtt == walk.rtte.addr`
result	`post: result.status == RMI_SUCCESS`
rtte_state	`post: walk.rtte.state == RTTE_AUX_DESTROYED`
rtt_state	`post: GranuleAt(walk.rtte.addr).state == GRAN_DELEGATED`

B4.5.50.4 Footprint

ID	Value
rtt_state	`GranuleAt(walk.rtte.addr).state`
rtte	`RttEntry(walk.rtt_addr, entry_idx)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.51 RMI_RTT_AUX_FOLD command

Destroys a homogeneous auxiliary RTT.

The RMI_RTT_AUX_FOLD command may initiate a Stateful RMI Operation.

See also:

- A5.6.6 *RTT folding*
- A10.3.1 *Auxiliary RTT*
- B4.5.49 *RMI_RTT_AUX_CREATE command*
- B4.5.50 *RMI_RTT_AUX_DESTROY command*

B4.5.51.1 Interface

B4.5.51.1.1 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

B4.5.51.1.2 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

B4.5.51.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
rtt	X1	63:0	Address	PA of the RTT which was destroyed

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.51.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

level_bound	`pre: (!RttLevelIsValid(realm, level)`
	`|| RttLevelIsStarting(realm, level))`
	`post: result.status == RMI_ERROR_INPUT`

ipa_align	`pre: !AddrIsRttLevelAligned(ipa, level - 1)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: !AddrIsProtected(ipa, realm)` `post: result.status == RMI_ERROR_INPUT`

index_bound	`pre: (realm.rtt_tree_per_plane == FEATURE_FALSE`
	`|| index == RMM_RTT_TREE_PRIMARY`
	`|| index > realm.num_aux_planes)`
	`post: result.status == RMI_ERROR_INPUT`

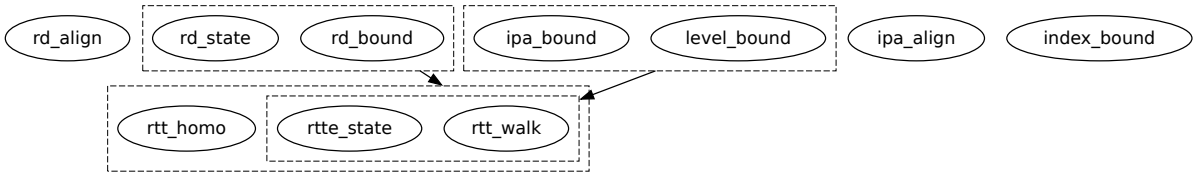
rtt_walk	`pre: walk.level < level - 1`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`&& result.data.level.level == walk.level)`

rtte_state	`pre: walk.rtte.state != RTTE_TABLE`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`&& result.data.level.level == walk.level)`

rtt_homo	`pre: !RttIsHomogeneous(RttAt(walk.rtte.addr))`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`&& result.data.level.level == level)`

B4.5.51.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state, rtt_homo]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.5.51.3 Success conditions

ID	Condition
rtt	`post: rtt == walk.rtte.addr`
result	`post: result.status == RMI_SUCCESS`
rtte_state	`post: walk.rtte.state == fold_pre.state`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rtte_addr	pre: fold_pre.state != RTTE_VOID && fold_pre.state != RTTE_UNMAPPED_NS post: walk.rtte.addr == fold_pre.addr
rtte_attr_prot	pre: fold_pre.state == RTTE_DATA post: (RttMemAttrEqual(walk.rtte, fold_pre, RTT_PROTECTED) && RttS2APEqual(walk.rtte, fold_pre, S2AP_INDIRECT))
rtte_attr_unprot	pre: fold_pre.state == RTTE_MAPPED_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	post: GranuleAt(walk.rtte.addr).state == GRAN_DELEGATED

B4.5.51.4 Footprint

ID	Value
rtt_state	GranuleAt(walk.rtte.addr).state
rtte	RttEntry(walk.rtt_addr, entry_idx)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.52 RMI_RTT_AUX_PROT_MAP command

Propagate mappings within a contiguous range of Protected IPA space from the primary RTT tree to an auxiliary RTT tree.

The RMI_RTT_AUX_PROT_MAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.13 *Create mappings within Protected IPA space in auxiliary RTT tree*
- A10.3.1 *Auxiliary RTT*
- B4.5.53 *RMI_RTT_AUX_PROT_UNMAP command*

B4.5.52.1 Interface

B4.5.52.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001FD`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttAuxMapFlags	Flags

B4.5.52.1.2 Context

The RMI_RTT_AUX_PROT_MAP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes

walk_pri	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of primary RTT walk from base of the target IPA range

walk_aux	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  flags.tree_index)`
			false	Result of auxiliary RTT walk from base of the target IPA range

B4.5.52.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{PRJZD} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.52.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

base_pri_align	`pre: !AddrIsAligned(`
	`base, RttLevelSize(walk_pri.level))`
	`post: result.status == RMI_ERROR_INPUT`

base_aux_align	`pre: (flags.block == RMI_RTT_AUX_BLOCK_NO_CREATE`
	`&& !AddrIsAligned(`
	`base, RttLevelSize(walk_aux.level)))`
	`post: result.status == RMI_ERROR_INPUT`

invalid_pri	`pre: (flags.invalid_pri == RMI_RTT_AUX_INVALID_PRI_STOP`
	`&& walk_pri.rtte.state == RTTE_VOID)`
	`post: result.status == RMI_ERROR_INPUT`

top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: !AddrRangeIsProtected(base, top, realm)` `post: result.status == RMI_ERROR_INPUT`

index_bound	`pre: (realm.rtt_tree_per_plane == FEATURE_FALSE`
	`|| flags.tree_index == RMM_RTT_TREE_PRIMARY`
	`|| flags.tree_index > realm.num_aux_planes)`
	`post: result.status == RMI_ERROR_INPUT`

pri_state	`pre: (walk_pri.rtte.state != RTTE_DATA`
	`&& walk_pri.rtte.state != RTTE_NARCH_DEV`
	`&& walk_pri.rtte.state != RTTE_ARCH_DEV)`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk_pri.level)`

pri_ram	`pre: (walk_pri.rtte.state == RTTE_DATA`
	`&& walk_pri.rtte.ripas != RIPAS_RAM)`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk_pri.level)`

pri_dev	`pre: (walk_pri.rtte.state == RTTE_NARCH_DEV`
	`&& walk_pri.rtte.ripas != RIPAS_DEV)`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk_pri.level)`

aux_destroyed	`pre: walk_aux.rtte.state == RTTE_AUX_DESTROYED`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`&& result.data.level.level == walk_aux.level)`

aux_size	`pre: (walk_aux.rtte.state == RTTE_VOID`
	`&& RttLevelSize(walk_aux.level) > size)`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`&& result.data.level.level == walk_aux.level)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.52.2.1 Failure condition ordering

The RMI_RTT_AUX_PROT_MAP command does not have any failure condition orderings.

B4.5.52.3 Success conditions

ID	Condition
state	`post: RttTreeRangeAllState( realm, flags.tree_index, base, out_top, RTTE_DATA)`
addr	`post: RttTreeRangeAllOaddrEqual( realm, RMM_RTT_TREE_PRIMARY, flags.tree_index, base, out_top)`
memattr	`post: RttTreeRangeAllMemAttrEqual( realm, RMM_RTT_TREE_PRIMARY, flags.tree_index, base, out_top)`
shareability	`post: RttTreeRangeAllShareabilityEqual( realm, RMM_RTT_TREE_PRIMARY, flags.tree_index, base, out_top)`
result	`post: result.status == RMI_SUCCESS`

B4.5.52.4 Footprint

The RMI_RTT_AUX_PROT_MAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.53 RMI_RTT_AUX_PROT_UNMAP command

Remove mappings within a contiguous range of Protected IPA space from an auxiliary RTT tree.

The RMI_RTT_AUX_PROT_UNMAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.14 *Remove mappings within Protected IPA space in auxiliary RTT tree*
- A10.3.1 *Auxiliary RTT*
- B4.5.52 *RMI_RTT_AUX_PROT_MAP command*

B4.5.53.1 Interface

B4.5.53.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001FE`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttAuxUnmapFlags	Flags

B4.5.53.1.2 Context

The RMI_RTT_AUX_PROT_UNMAP command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	`RealmAt(rd)`	true	Realm
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes
progress	UInt64	`UInt(out_top) - UInt(base)`	false	Size of IPA range which has been unmapped

walk_pri	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of primary RTT walk from base of the target IPA range

walk_aux	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  flags.tree_index)`
			false	Result of auxiliary RTT walk from base of the target IPA range

B4.5.53.1.3 Output values

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

I_JHGCD If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.53.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

base_align	`pre: !AddrIsAligned(`
	`  base, RttLevelSize(walk_pri.level))`
	`post: result.status == RMI_ERROR_INPUT`

top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: !AddrRangeIsProtected(base, top, realm)` `post: result.status == RMI_ERROR_INPUT`

index_bound	`pre: (realm.rtt_tree_per_plane == FEATURE_FALSE`
	`  || flags.tree_index == RMM_RTT_TREE_PRIMARY`
	`  || flags.tree_index > realm.num_aux_planes)`
	`post: result.status == RMI_ERROR_INPUT`

aux_state	`pre: walk_aux.rtte.state != RTTE_DATA`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`  && result.data.level.level == walk_aux.level)`

aux_size	`pre: (walk_aux.rtte.state == RTTE_VOID`
	`  && RttLevelSize(walk_aux.level) > size)`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`  && result.data.level.level == walk_aux.level)`

B4.5.53.2.1 Failure condition ordering

The RMI_RTT_AUX_PROT_UNMAP command does not have any failure condition orderings.

B4.5.53.3 Success conditions

ID	Condition

state	`post: RttTreeRangeAllState(`
	`  realm, flags.tree_index,`
	`  base, out_top, RTTE_VOID)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
ripas	`post: RealmIpaRangeAllRipasIf(realm_pre, realm, base, out_top, RIPAS_RAM, RIPAS_DESTROYED)`
result	`post: result.status == RMI_SUCCESS`

B4.5.53.4 Footprint

The RMI_RTT_AUX_PROT_UNMAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.54 RMI_RTT_AUX_UNPROT_MAP command

Propagate mappings within a contiguous range of Unprotected IPA space from the primary RTT tree to an auxiliary RTT tree.

The RMI_RTT_AUX_UNPROT_MAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.15 *Create mappings within Unprotected IPA space in auxiliary RTT tree*
- A10.3.1 *Auxiliary RTT*
- B4.5.55 *RMI_RTT_AUX_UNPROT_UNMAP command*

B4.5.54.1 Interface

B4.5.54.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001FF`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttAuxMapFlags	Flags

B4.5.54.1.2 Context

The RMI_RTT_AUX_UNPROT_MAP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes

walk_pri	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of primary RTT walk from base of the target IPA range

walk_aux	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  flags.tree_index)`
			false	Result of auxiliary RTT walk from base of the target IPA range

B4.5.54.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

I_{WDFHK} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.54.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

base_align	`pre: !AddrIsAligned(`
	`base, RttLevelSize(walk_pri.level))`
	`post: result.status == RMI_ERROR_INPUT`

top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: AddrIsProtected(base, realm)` `post: result.status == RMI_ERROR_INPUT`

index_bound	`pre: (realm.rtt_tree_per_plane == FEATURE_FALSE`
	`|| flags.tree_index == RMM_RTT_TREE_PRIMARY`
	`|| flags.tree_index > realm.num_aux_planes)`
	`post: result.status == RMI_ERROR_INPUT`

pri_state	`pre: walk_pri.rtte.state != RTTE_MAPPED_NS`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk_pri.level)`

aux_size	`pre: (walk_aux.rtte.state == RTTE_VOID`
	`&& RttLevelSize(walk_aux.level) > size)`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`&& result.data.level.level == walk_aux.level)`

B4.5.54.2.1 Failure condition ordering

The RMI_RTT_AUX_UNPROT_MAP command does not have any failure condition orderings.

B4.5.54.3 Success conditions

ID	Condition

state	`post: RttTreeRangeAllState(`
	`realm, flags.tree_index,`
	`base, out_top, RTTE_MAPPED_NS)`

addr	`post: RttTreeRangeAllOaddrEqual(`
	`realm, RMM_RTT_TREE_PRIMARY, flags.tree_index,`
	`base, out_top)`

memattr	`post: RttTreeRangeAllMemAttrEqual(`
	`realm, RMM_RTT_TREE_PRIMARY, flags.tree_index,`
	`base, out_top)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
shareability	`post: RttTreeRangeAllShareabilityEqual( realm, RMM_RTT_TREE_PRIMARY, flags.tree_index, base, out_top)`
result	`post: result.status == RMI_SUCCESS`

B4.5.54.4 Footprint

The RMI_RTT_AUX_UNPROT_MAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.55 RMI_RTT_AUX_UNPROT_UNMAP command

Remove mappings within a contiguous range of Unprotected IPA space from an auxiliary RTT tree.

The RMI_RTT_AUX_UNPROT_UNMAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.16 *Remove mappings within Unprotected IPA space in auxiliary RTT tree*
- A10.3.1 *Auxiliary RTT*
- B4.5.54 *RMI_RTT_AUX_UNPROT_MAP command*

B4.5.55.1 Interface

B4.5.55.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000200`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttAuxUnmapFlags	Flags

B4.5.55.1.2 Context

The RMI_RTT_AUX_UNPROT_UNMAP command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	`RealmAt(rd)`	true	Realm
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes
progress	UInt64	`UInt(out_top) - UInt(base)`	false	Size of IPA range which has been unmapped

walk_pri	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of primary RTT walk from base of the target IPA range

walk_aux	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  flags.tree_index)`
			false	Result of auxiliary RTT walk from base of the target IPA range

B4.5.55.1.3 Output values

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

I_{RPJLW} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.55.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

base_align	`pre: !AddrIsAligned(`
	`  base, RttLevelSize(walk_pri.level))`
	`post: result.status == RMI_ERROR_INPUT`

top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: AddrIsProtected(base, realm)` `post: result.status == RMI_ERROR_INPUT`

index_bound	`pre: (realm.rtt_tree_per_plane == FEATURE_FALSE`
	`  || flags.tree_index == RMM_RTT_TREE_PRIMARY`
	`  || flags.tree_index > realm.num_aux_planes)`
	`post: result.status == RMI_ERROR_INPUT`

aux_state	`pre: walk_aux.rtte.state != RTTE_MAPPED_NS`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`  && result.data.level.level == walk_aux.level)`

aux_size	`pre: (walk_aux.rtte.state == RTTE_UNMAPPED_NS`
	`  && RttLevelSize(walk_aux.level) > size)`
	`post: (result.status == RMI_ERROR_RTT_AUX`
	`  && result.data.level.level == walk_aux.level)`

B4.5.55.2.1 Failure condition ordering

The RMI_RTT_AUX_UNPROT_UNMAP command does not have any failure condition orderings.

B4.5.55.3 Success conditions

ID	Condition

state	`post: RttTreeRangeAllState(`
	`  realm, flags.tree_index,`
	`  base, out_top, RTTE_UNMAPPED_NS)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
result	`post: result.status == RMI_SUCCESS`

B4.5.55.4 Footprint

The RMI_RTT_AUX_UNPROT_UNMAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.56 RMI_RTT_CREATE command

Creates a primary RTT.

The RMI_RTT_CREATE command may initiate a Stateful RMI Operation.

See also:

- A5.6 *Realm Translation Table*
- A5.6.7 *RTT unfolding*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.64 *RMI_RTT_FOLD command*

B4.5.56.1 Interface

B4.5.56.1.1 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

B4.5.56.1.2 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

B4.5.56.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.56.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

level_bound	`pre: (!RttLevelIsValid(realm, level)`
	`|| RttLevelIsStarting(realm, level))`
	`post: result.status == RMI_ERROR_INPUT`

ipa_align	`pre: !AddrIsRttLevelAligned(ipa, level - 1)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: UInt(ipa) >= (2 ^ realm.ipa_width)` `post: result.status == RMI_ERROR_INPUT`
rtt_align	`pre: !AddrIsRmiGranuleAligned(rtt)` `post: result.status == RMI_ERROR_INPUT`
rtt_bound	`pre: !PaIsDelegableConventionalFine(rtt)` `post: result.status == RMI_ERROR_INPUT`
rtt_state	`pre: GranuleAt(rtt).state != GRAN_DELEGATED` `post: result.status == RMI_ERROR_INPUT`

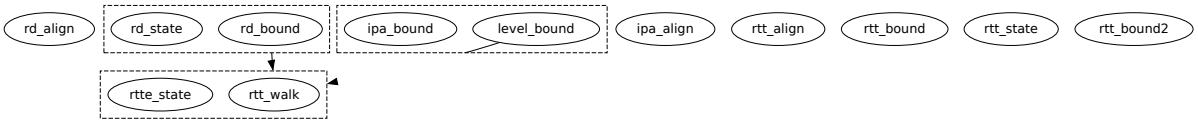
rtt_bound2	`pre: ((realm.feat_lpa2 == FEATURE_FALSE)`
	`&& (UInt(rtt) >= 2^48))`
	`post: result.status == RMI_ERROR_INPUT`

rtt_walk	`pre: walk.level < level - 1`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

rtte_state	`pre: walk.rtte.state == RTTE_TABLE`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

B4.5.56.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.5.56.3 Success conditions

ID	Condition
rtt_state	`post: GranuleAt(rtt).state == GRAN_RTT`
rtte_addr	`post: walk.rtte.addr == rtt`
result	`post: result.status == RMI_SUCCESS`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rtte_state	`post: walk.rtte.state == RTTE_TABLE`
rtte_c_ripas	`pre: AddrIsProtected(ipa, realm)` `post: RttAllEntriesRipas(RttAt(rtt), rtte_pre.ripas)`
rtte_c_state	`post: RttAllEntriesState(RttAt(rtt), rtte_pre.state)`

rtte_c_addr	`pre: rtte_pre.state != RTTE_VOID`
	`&& rtte_pre.state != RTTE_UNMAPPED_NS`
	`post: RttAllEntriesContiguous(RttAt(rtt), rtte_pre.addr, level)`

rtte_c_mem_attr	`pre: rtte_pre.state != RTTE_VOID`
	`&& rtte_pre.state != RTTE_UNMAPPED_NS`
	`post: RttAllEntriesMemAttr(RttAt(rtt), rtte_pre)`

rtte_c_s2ap	`pre: AddrIsProtected(ipa, realm)` `post: RttAllEntriesS2AP(RttAt(rtt), rtte_pre)`

B4.5.56.4 Footprint

ID	Value
rtt_state	`GranuleAt(rtt).state`
rtte	`RttEntryAt(RttAt(walk.rtt_addr), entry_idx)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.57 RMI_RTT_DATA_MAP command

Create mappings to conventional memory within a target Protected IPA range.

The RMI_RTT_DATA_MAP command may initiate a Stateful RMI Operation.

See also:

- A2.3.8 *Granule wiping*
- A5.3.5 *Create mappings from Protected IPA space to wiped conventional memory*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- D1.5.1 *Add memory to Active Realm flow*

B4.5.57.1 Interface

B4.5.57.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F5`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttProtMapFlags	Flags
oaddr	X5	63:0	RmiAddrSetDesc	Output address set descriptor. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this describes a contiguous PA range which will be mapped into the target IPA range. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the PA of a Granule that holds an RMI Address Range List. This describes a list of PA regions which will be mapped into the target IPA range.

DRAFT

B4.5.57.1.2 Context

The RMI_RTT_DATA_MAP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes
progress	UInt64	`UInt(out_top) - UInt(base)`	false	Size of IPA range which has been mapped

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Value	Before	Description

walk	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of RTT walk from base of the target IPA range

oaddr_first	Address	`AddrSetEntry(`
		`  oaddr,`
		`  flags.oaddr_type,`
		`  0)`
			false	PA of start of the output address set

region	RmmTrackingRegion	`TrackingRegionAt(` `  oaddr_first)`	false	Tracking region containing start of the output address set

B4.5.57.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

I_{PBSPY} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.57.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
base_align	`pre: !AddrIsRmiGranuleAligned(base)` `post: result.status == RMI_ERROR_INPUT`
top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: !AddrRangeIsProtected(base, top, realm)` `post: result.status == RMI_ERROR_INPUT`

oaddr_single_al ign	`pre: (flags.oaddr_type == RMI_ADDR_TYPE_SINGLE`
	`  && !AddrIsRmiGranuleAligned(oaddr.data.single.addr))`
	`post: result.status == RMI_ERROR_INPUT`

oaddr_list_alig n	`pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST`
	`  && !AddrIsAligned(oaddr.data.list_addr.addr, 8))`
	`post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
oaddr_type	pre: (flags.oaddr_type != RMI_ADDR_TYPE_SINGLE && flags.oaddr_type != RMI_ADDR_TYPE_LIST) post: result.status == RMI_ERROR_INPUT
oaddr_list_pas	pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST && !NonSecureAccessPermitted(oaddr.data.list_addr.addr)) post: result.status == RMI_ERROR_INPUT
rtte_state	pre: walk.rtte.state != RTTE_VOID post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
rtte_size	pre: (walk.rtte.state == RTTE_VOID && RttLevelSize(walk.level) > size) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
trk_untracked	pre: !TrackingRegionIsTracked(region) post: result.status == RMI_ERROR_TRACKING
trk_gran	pre: (TrackingRegionIsTracked(region) && TrackingRegionGranularity(region) > size) post: result.status == RMI_ERROR_TRACKING
gran_state	pre: GranuleAt(oaddr_first).state != GRAN_DELEGATED post: result.status == RMI_ERROR_INPUT
dpt_gran	pre: (realm.feat_ats == FEATURE_TRUE && !DptEntryCanDescribe(oaddr_first, size)) post: result.status == RMI_ERROR_DPT

B4.5.57.2.1 Failure condition ordering

The RMI_RTT_DATA_MAP command does not have any failure condition orderings.

B4.5.57.3 Success conditions

ID	Condition
state	post: RttTreeRangeAllState(realm, RMM_RTT_TREE_PRIMARY, base, out_top, RTTE_DATA)
addr_contig	pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE post: RttTreeRangeAllOaddrContig(realm, RMM_RTT_TREE_PRIMARY, base, RmiAddrRangeDescDecode(oaddr.data.single).base, progress)
addr_list	pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST post: RttTreeRangeAllOaddrList(realm, RMM_RTT_TREE_PRIMARY, base, oaddr.data.list_addr.addr, progress)
mematttr	post: RttTreeRangeAllMemAttr(realm, RMM_RTT_TREE_PRIMARY, base, out_top, MEMATTR_CACHEABLE)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
shareability	`post: RttTreeRangeAllShareability( realm, RMM_RTT_TREE_PRIMARY, base, out_top, SHAREABILITY_INNER)`
gran_contents	`post: Contents of first size bytes of output address set are wiped.`
gran_state_contig	`pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE` `post: GranulesAllState( RmiAddrRangeDescDecode(oaddr.data.single).base, size, GRAN_DATA)`
gran_state_list	`pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST` `post: GranulesAllStateList( oaddr.data.list_addr.addr, size, GRAN_DATA)`
result	`post: result.status == RMI_SUCCESS`

B4.5.57.4 Footprint

The RMI_RTT_DATA_MAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.58 RMI_RTT_DATA_MAP_INIT command

Create a mapping from Protected IPA space to conventional memory, copying contents from a Non-secure Granule provided by the caller.

The RMI_RTT_DATA_MAP_INIT command may initiate a Stateful RMI Operation.

See also:

- A5.3.4 *Mapping initial Realm image in Protected IPA space*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- D1.2.3 *Initialize memory of New Realm flow*

B4.5.58.1 Interface

B4.5.58.1.1 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

DRAFT

B4.5.58.1.2 Context

The RMI_RTT_DATA_MAP_INIT 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

B4.5.58.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

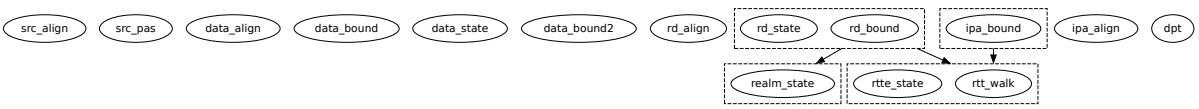
Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.58.2 Failure conditions

ID	Condition
src_align	pre: !AddrIsRmiGranuleAligned(src) post: result.status == RMI_ERROR_INPUT
src_pas	pre: !NonSecureAccessPermitted(src) post: result.status == RMI_ERROR_INPUT
data_align	pre: !AddrIsRmiGranuleAligned(data) post: result.status == RMI_ERROR_INPUT
data_bound	pre: !PaIsDelegableConventionalFine(data) post: result.status == RMI_ERROR_INPUT
data_state	pre: GranuleAt(data).state != GRAN_DELEGATED post: result.status == RMI_ERROR_INPUT
data_bound2	pre: ((realm.feat_lpa2 == FEATURE_FALSE) && (UInt(data) >= 2^48)) post: result.status == RMI_ERROR_INPUT
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
ipa_align	pre: !AddrIsRmiGranuleAligned(ipa) post: result.status == RMI_ERROR_INPUT
ipa_bound	pre: !AddrIsProtected(ipa, realm_pre) post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm_pre.state != REALM_NEW post: result.status == RMI_ERROR_REALM
rtt_walk	pre: walk.level < RMM_RTT_PAGE_LEVEL post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
rtte_state	pre: walk.rtte.state != RTTE_VOID post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
dpt	pre: Missing failure conditions: - ATS is enabled for the Realm and either: - Any ATS-capable PSMMU has not been activated - L1DPT is missing post: result.status == RMI_ERROR_INPUT

B4.5.58.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[ipa_bound] < [rtt_walk, rtte_state]
```



Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.58.3 Success conditions

ID	Condition
data_state	`post: GranuleAt(data).state == GRAN_DATA`
data_content	`post: Contents of target Granule are copied from source Granule.`
rtte_ripas	`post: walk.rtte.ripas == RIPAS_RAM`
rtte_addr	`post: walk.rtte.addr == data`
rtte_memattr	`post: walk.rtte.attr_prot == MEMATTR_CACHEABLE`
rtte_sh	`post: walk.rtte.sh == SHAREABILITY_INNER`
rim	`post: realm.rim == RimExtendData(realm_pre, ipa, data, flags)`
result	`post: result.status == RMI_SUCCESS`
rtte_state	`post: walk.rtte.state == RTTE_DATA`

B4.5.58.4 RMI_RTT_DATA_MAP_INIT extension of RIM

On successful execution of RMI_RTT_DATA_MAP_INIT, 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*
- B3.106 *RimExtendData function*
- C2.25 *RmmMeasurementDescriptorData type*

B4.5.58.5 Footprint

ID	Value
data_state	`GranuleAt(data).state`
rim	`realm.rim`
rtte	`RttEntryAt(RttAt(walk.rtt_addr), entry_idx)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.59 RMI_RTT_DATA_UNMAP command

Removes mappings to conventional memory within a target Protected IPA range.

The RMI_RTT_DATA_UNMAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.6 *Remove mappings from Protected IPA space to conventional memory*
- B4.5.57 *RMI_RTT_DATA_MAP command*
- D1.5.1 *Add memory to Active Realm flow*

B4.5.59.1 Interface

B4.5.59.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F6`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttUnmapFlags	Flags
oaddr	X5	63:0	RmiAddrSetDesc	Output address set descriptor. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this SBZ. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the PA of a Granule that holds an RMI Address Range List. This describes a list of PA regions which have been unmapped from the target IPA range.

DRAFT

B4.5.59.1.2 Context

The RMI_RTT_DATA_UNMAP command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	`RealmAt(rd)`	true	Realm
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes
progress	UInt64	`UInt(out_top) - UInt(base)`	false	Size of IPA range which has been unmapped

walk_pre	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			true	Result of RTT walk from base of the target IPA range

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Value	Before	Description
oaddr_first	Address	`walk_pre.rtte.addr`	false	PA of start of the output address set
region	RmmTrackingRegion	`TrackingRegionAt(` `  oaddr_first)`	false	Tracking region containing start of the output address set

walk	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of RTT walk from base of the target IPA range

B4.5.59.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been unmapped
out_range	X2	63:0	RmiAddrRangeDesc	Output address range. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this describes a contiguous PA range which has been unmapped from the target IPA range. If flags.oaddr_type != RMI_ADDR_TYPE_SINGLE then this value is zero.
out_count	X3	63:0	UInt64	Number of entries in output address list. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the number of entries which have been written to the output address list. If flags.oaddr_type != RMI_ADDR_TYPE_LIST then this value is zero.

DRAFT

I_{VTKMV} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.59.2 Failure conditions

ID	Condition
rd_align	`pre:  !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre:  !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre:  GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
base_align	pre: !AddrIsRmiGranuleAligned(base) post: result.status == RMI_ERROR_INPUT
top_align	pre: !AddrIsRmiGranuleAligned(top) post: result.status == RMI_ERROR_INPUT
size_valid	pre: UInt(top) <= UInt(base) post: result.status == RMI_ERROR_INPUT
ipa_bound	pre: !AddrRangeIsProtected(base, top, realm) post: result.status == RMI_ERROR_INPUT
oaddr_align	pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST && !AddrIsAligned(oaddr.data.list_addr.addr, 8)) post: result.status == RMI_ERROR_INPUT
oaddr_list_pas	pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST && !NonSecureAccessPermitted(oaddr.data.list_addr.addr)) post: result.status == RMI_ERROR_INPUT
rtte_size	pre: (walk.rtte.state == RTTE_DATA && RttLevelSize(walk.level) > size) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
trk_gran	pre: (TrackingRegionIsTracked(region) && TrackingRegionGranularity(region) > size) post: result.status == RMI_ERROR_TRACKING
dpt_gran	pre: (realm.feat_ats == FEATURE_TRUE && !DptEntryCanDescribe(oaddr_first, size)) post: result.status == RMI_ERROR_DPT

B4.5.59.2.1 Failure condition ordering

The RMI_RTT_DATA_UNMAP command does not have any failure condition orderings.

B4.5.59.3 Success conditions

ID	Condition
state	post: RttTreeRangeAllState(realm, RMM_RTT_TREE_PRIMARY, base, out_top, RTTE_VOID)
ripas	post: RealmIpaRangeAllRipasIf(realm_pre, realm, base, out_top, RIPAS_RAM, RIPAS_DESTROYED)
addr_single	pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE post: out_range.data.addr == walk_pre.rtte.addr
addr_list	pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST post: RttTreeRangeAllOaddrList(realm, RMM_RTT_TREE_PRIMARY, base, oaddr.data.list_addr.addr, progress)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
gran_state_contig	pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE post: GranulesAllState(RmiAddrRangeDescDecode(oaddr.data.single).base, progress, GRAN_DELEGATED)
gran_state_list	pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST post: GranulesAllStateList(oaddr.data.list_addr.addr, progress, GRAN_DELEGATED)
result	post: result.status == RMI_SUCCESS

B4.5.59.4 Footprint

The RMI_RTT_DATA_UNMAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.60 RMI_RTT_DESTROY command

Destroys a primary RTT.

The RMI_RTT_DESTROY command may initiate a Stateful RMI Operation.

See also:

- A5.6 *Realm Translation Table*
- A5.6.9 *RTT destruction*
- B4.5.56 *RMI_RTT_CREATE command*
- B4.5.64 *RMI_RTT_FOLD command*

B4.5.60.1 Interface

B4.5.60.1.1 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

B4.5.60.1.2 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

B4.5.60.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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	RMI_SUCCESS	ipa	Before execution: RTTE_TABLE After execution: RTTE_VOID and RIPAS is RIPAS_DESTROYED
Missing RTT	(RMI_ERROR_RTT, < level)	ipa	RTTE_VOID or RTTE_UNMAPPED_NS
Block mapping at lower level	(RMI_ERROR_RTT, < level)	== ipa	RTTE_DATA or RTTE_MAPPED_NS
Live RTT at target level	(RMI_ERROR_RTT, level)	== ipa	RTTE_TABLE
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*

B4.5.60.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

level_bound	`pre: (!RttLevelIsValid(realm, level)`
	`  || RttLevelIsStarting(realm, level))`
	`post: result.status == RMI_ERROR_INPUT`

ipa_align	`pre: !AddrIsRttLevelAligned(ipa, level - 1)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: UInt(ipa) >= (2 ^ realm.ipa_width)` `post: result.status == RMI_ERROR_INPUT`

rtt_walk	`pre: walk.level < level - 1`
	`post: (result.status == RMI_ERROR_RTT`
	`  && result.data.level.level == walk.level`
	`  && top == walk_top)`

rtte_state	`pre: walk.rtte.state != RTTE_TABLE`
	`post: (result.status == RMI_ERROR_RTT`
	`  && result.data.level.level == walk.level`
	`  && top == walk_top)`

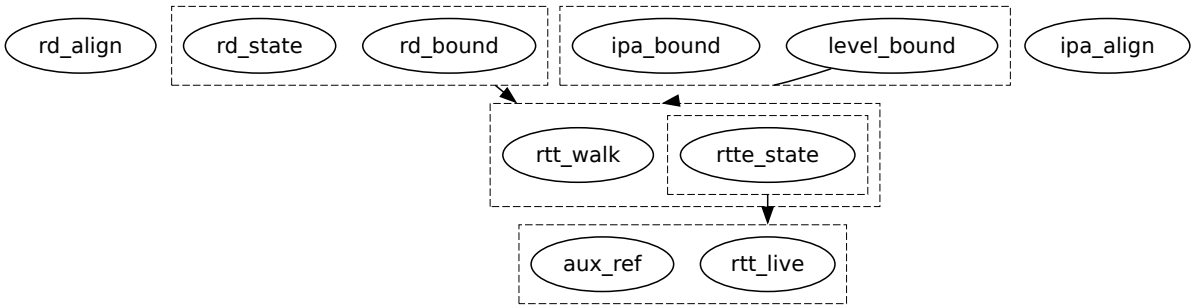
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rtt_live	pre: RttIsLive(RttAt(walk.rtte.addr)) post: (result.status == RMI_ERROR_RTT && result.data.level.level == level && top == ipa)
aux_ref	pre: AddrIsAuxRef(ipa, realm) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)

B4.5.60.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state]
[rtte_state] < [rtt_live, aux_ref]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.5.60.3 Success conditions

ID	Condition
rtt	post: rtt == walk.rtte.addr
top	post: top == walk_top
result	post: result.status == RMI_SUCCESS
state_prot	pre: AddrIsProtected(ipa, realm) post: walk.rtte.state == RTTE_VOID
ripas	pre: AddrIsProtected(ipa, realm) post: walk.rtte.ripas == RIPAS_DESTROYED
state_unprot	pre: !AddrIsProtected(ipa, realm) post: walk.rtte.state == RTTE_UNMAPPED_NS
rtt_state	post: GranuleAt(walk.rtte.addr).state == GRAN_DELEGATED

B4.5.60.4 Footprint

ID	Value
rtt_state	GranuleAt(walk.rtte.addr).state
rtte	RttEntryAt(RttAt(walk.rtt_addr), entry_idx)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.61 RMI_RTT_DEV_MAP command

Create mappings to device memory within a target Protected IPA range.

The RMI_RTT_DEV_MAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.7 *Create mappings from Protected IPA space to device memory*
- Chapter A9 *Realm device assignment*
- B4.5.62 *RMI_RTT_DEV_UNMAP command*

B4.5.61.1 Interface

B4.5.61.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F7`
rd	X1	63:0	Address	PA of the RD for the target Realm
vdev_ptr	X2	63:0	Address	PA of the VDEV
base	X3	63:0	Address	Base of the target IPA range
top	X4	63:0	Address	Top of the target IPA range
flags	X5	63:0	RmiRttProtMapFlags	Flags
oaddr	X6	63:0	RmiAddrSetDesc	Output address set descriptor. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this describes a contiguous PA range which will be mapped into the target IPA range. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the PA of a Granule that holds an RMI Address Range List. This describes a list of PA regions which will be mapped into the target IPA range.

DRAFT

B4.5.61.1.2 Context

The RMI_RTT_DEV_MAP command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
realm	RmmRealm	`RealmAt(rd)`	false	Realm
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes
progress	UInt64	`UInt(out_top) - UInt(base)`	false	Size of IPA range which has been mapped

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Value	Before	Description

walk	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of RTT walk from base of the target IPA range

oaddr_first	Address	`AddrSetEntry(`
		`  oaddr,`
		`  flags.oaddr_type,`
		`  0)`
			false	PA of start of the output address set

region	RmmTrackingRegion	`TrackingRegionAt(` `  oaddr_first)`	false	Tracking region containing start of the output address set

B4.5.61.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

I_{SPLWG} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.61.2 Failure conditions

ID	Condition
rd_align	`pre:  !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre:  !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre:  GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
vdev_align	`pre:  !AddrIsRmiGranuleAligned(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_bound	`pre:  !PaIsTracked(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_gran_state	`pre:  GranuleAt(vdev_ptr).state != GRAN_VDEV` `post: result.status == RMI_ERROR_INPUT`
vdev_realm	`pre:  vdev.realm != rd` `post: result.status == RMI_ERROR_INPUT`
base_align	`pre:  !AddrIsRmiGranuleAligned(base)` `post: result.status == RMI_ERROR_INPUT`
top_align	`pre:  !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre:  UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
ipa_bound	pre: !AddrRangeIsProtected(base, top, realm) post: result.status == RMI_ERROR_INPUT
oaddr_single_al ign	pre: (flags.oaddr_type == RMI_ADDR_TYPE_SINGLE && !AddrIsRmiGranuleAligned(oaddr.data.single.addr)) post: result.status == RMI_ERROR_INPUT
oaddr_list_alig n	pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST && !AddrIsAligned(oaddr.data.list_addr.addr, 8)) post: result.status == RMI_ERROR_INPUT
oaddr_type	pre: (flags.oaddr_type != RMI_ADDR_TYPE_SINGLE && flags.oaddr_type != RMI_ADDR_TYPE_LIST) post: result.status == RMI_ERROR_INPUT
oaddr_list_pas	pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST && !NonSecureAccessPermitted(oaddr.data.list_addr.addr)) post: result.status == RMI_ERROR_INPUT
rtte_state	pre: walk.rtte.state != RTTE_VOID post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
rtte_size	pre: (walk.rtte.state == RTTE_VOID && RttLevelSize(walk.level) > size) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
rtte_ripas	pre: walk.rtte.ripas == RIPAS_RAM post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
trk_untracked	pre: !TrackingRegionIsTracked(region) post: result.status == RMI_ERROR_TRACKING
trk_gran	pre: (TrackingRegionIsTracked(region) && TrackingRegionGranularity(region) > size) post: result.status == RMI_ERROR_TRACKING
gran_state	pre: GranuleAt(oaddr_first).state != GRAN_DELEGATED post: result.status == RMI_ERROR_INPUT
dpt_gran	pre: (realm.feat_ats == FEATURE_TRUE && !DptEntryCanDescribe(oaddr_first, size)) post: result.status == RMI_ERROR_DPT
oaddr_bound	pre: !VdevAddrInRange(oaddr_first, vdev) post: result.status == RMI_ERROR_INPUT

B4.5.61.2.1 Failure condition ordering

The RMI_RTT_DEV_MAP command does not have any failure condition orderings.

B4.5.61.3 Success conditions

ID	Condition
state	post: RttTreeRangeAllState(realm, RMM_RTT_TREE_PRIMARY, base, out_top, RTTE_NARCH_DEV)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
addr_contig	pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE post: RttTreeRangeAllOaddrContig(realm, RMM_RTT_TREE_PRIMARY, base, RmiAddrRangeDescDecode(oaddr.data.single).base, progress)
addr_list	pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST post: RttTreeRangeAllOaddrList(realm, RMM_RTT_TREE_PRIMARY, base, oaddr.data.list_addr.addr, progress)
mematttr_ncoh	pre: AddrSetAllDelegableNonCohDevMem(oaddr, flags.oaddr_type, progress) post: RttTreeRangeAllMemAttr(realm, RMM_RTT_TREE_PRIMARY, base, out_top, MEMATTR_NON_CACHEABLE)
mematttr_coh	pre: AddrSetAllDelegableCohDevMem(oaddr, flags.oaddr_type, progress) post: RttTreeRangeAllMemAttr(realm, RMM_RTT_TREE_PRIMARY, base, out_top, MEMATTR_PASSTHROUGH)
shareability_ncoh	pre: AddrSetAllDelegableNonCohDevMem(oaddr, flags.oaddr_type, progress) post: RttTreeRangeAllShareability(realm, RMM_RTT_TREE_PRIMARY, base, out_top, SHAREABILITY_OUTER)
shareability_coh	pre: AddrSetAllDelegableCohDevMem(oaddr, flags.oaddr_type, progress) post: RttTreeRangeAllShareability(realm, RMM_RTT_TREE_PRIMARY, base, out_top, SHAREABILITY_INNER)
gran_state_contig	pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE post: GranulesAllState(RmiAddrRangeDescDecode(oaddr.data.single).base, progress, GRAN_DEV)
gran_state_list	pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST post: GranulesAllStateList(oaddr.data.list_addr.addr, progress, GRAN_DEV)
result	post: result.status == RMI_SUCCESS

B4.5.61.4 Footprint

The RMI_RTT_DEV_MAP command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.62 RMI_RTT_DEV_UNMAP command

Removes mappings to device memory within a target Protected IPA range.

The RMI_RTT_DEV_UNMAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.8 *Remove mappings from Protected IPA space to device memory*
- Chapter A9 *Realm device assignment*
- B4.5.61 *RMI_RTT_DEV_MAP command*

B4.5.62.1 Interface

B4.5.62.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001F8`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttUnmapFlags	Flags
oaddr	X5	63:0	RmiAddrSetDesc	Output address set descriptor. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this SBZ. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the PA of a Granule that holds an RMI Address Range List. This describes a list of PA regions which have been unmapped from the target IPA range.

DRAFT

B4.5.62.1.2 Context

The RMI_RTT_DEV_UNMAP command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
realm_pre	RmmRealm	`RealmAt(rd)`	true	Realm
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes
progress	UInt64	`UInt(out_top) - UInt(base)`	false	Size of IPA range which has been unmapped

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Value	Before	Description

walk_pre	RmmRttWalkResult	`RttWalk(`
		`realm, base,`
		`RMM_RTT_PAGE_LEVEL,`
		`RMM_RTT_TREE_PRIMARY)`
			true	Result of RTT walk from base of the target IPA range

oaddr_first	Address	`walk_pre.rtte.addr`	false	PA of start of the output address set
region	RmmTrackingRegion	`TrackingRegionAt(` `oaddr_first)`	false	Tracking region containing start of the output address set

walk	RmmRttWalkResult	`RttWalk(`
		`realm, base,`
		`RMM_RTT_PAGE_LEVEL,`
		`RMM_RTT_TREE_PRIMARY)`
			false	Result of RTT walk from base of the target IPA range

B4.5.62.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been unmapped
out_range	X2	63:0	RmiAddrRangeDesc	Output address range. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this describes a contiguous PA range which has been unmapped from the target IPA range. If flags.oaddr_type != RMI_ADDR_TYPE_SINGLE then this value is zero.
out_count	X3	63:0	UInt64	Number of entries in output address list. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the number of entries which have been written to the output address list. If flags.oaddr_type != RMI_ADDR_TYPE_LIST then this value is zero.

DRAFT

I_{DBCPL} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.62.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
base_align	pre: !AddrIsRmiGranuleAligned(base) post: result.status == RMI_ERROR_INPUT
top_align	pre: !AddrIsRmiGranuleAligned(top) post: result.status == RMI_ERROR_INPUT
size_valid	pre: UInt(top) <= UInt(base) post: result.status == RMI_ERROR_INPUT
ipa_bound	pre: !AddrRangeIsProtected(base, top, realm) post: result.status == RMI_ERROR_INPUT
oaddr_align	pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST && !AddrIsAligned(oaddr.data.list_addr.addr, 8)) post: result.status == RMI_ERROR_INPUT
oaddr_list_pas	pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST && !NonSecureAccessPermitted(oaddr.data.list_addr.addr)) post: result.status == RMI_ERROR_INPUT
rtte_state	pre: walk.rtte.state != RTTE_NARCH_DEV post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
rtte_size	pre: (walk.rtte.state == RTTE_NARCH_DEV && RttLevelSize(walk.level) > size) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
trk_gran	pre: (TrackingRegionIsTracked(region) && TrackingRegionGranularity(region) > size) post: result.status == RMI_ERROR_TRACKING
dpt_gran	pre: (realm.feat_ats == FEATURE_TRUE && !DptEntryCanDescribe(oaddr_first, size)) post: result.status == RMI_ERROR_DPT

B4.5.62.2.1 Failure condition ordering

The RMI_RTT_DEV_UNMAP command does not have any failure condition orderings.

B4.5.62.3 Success conditions

ID	Condition
state	post: RttTreeRangeAllState(realm, RMM_RTT_TREE_PRIMARY, base, out_top, RTTE_VOID)
ripas	post: RealmIpaRangeAllRipasIf(realm_pre, realm, base, out_top, RIPAS_DEV, RIPAS_DESTROYED)
addr_single	pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE post: out_range.data.addr == walk_pre.rtte.addr

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
addr_list	pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST post: RttTreeRangeAllOaddrList(realm, RMM_RTT_TREE_PRIMARY, base, oaddr.data.list_addr.addr, progress)
gran_state_contig	pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE post: GranulesAllState(RmiAddrRangeDescDecode(oaddr.data.single).base, progress, GRAN_DELEGATED)
gran_state_list	pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST post: GranulesAllStateList(oaddr.data.list_addr.addr, progress, GRAN_DELEGATED)
result	post: result.status == RMI_SUCCESS

B4.5.62.4 Footprint

The RMI_RTT_DEV_UNMAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.63 RMI_RTT_DEV_VALIDATE command

Completes a request made by the Realm to validate mappings to device memory from a target IPA range.

The RMI_RTT_DEV_VALIDATE command may initiate a Stateful RMI Operation.

See also:

- A5.5 *VDEV mapping validation*
- A9.6.2 *Realm validation of device memory mappings*
- B4.3.5 *Range RMI operations*

B4.5.63.1 Interface

B4.5.63.1.1 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
pdev_ptr	X3	63:0	Address	PA of the PDEV
vdev_ptr	X4	63:0	Address	PA of the VDEV
base	X5	63:0	Address	Base of target IPA region
top	X6	63:0	Address	Top of target IPA region

DRAFT

B4.5.63.1.2 Context

The RMI_RTT_DEV_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
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV
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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.63.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range whose RIPAS was modified

I_{KRHBB} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.63.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
rec_align	`pre: !AddrIsRmiGranuleAligned(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_bound	`pre: !PaIsTracked(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_gran_state	`pre: GranuleAt(rec_ptr).state != GRAN_REC` `post: result.status == RMI_ERROR_INPUT`
rec_state	`pre: rec.state == REC_RUNNING` `post: result.status == RMI_ERROR_REC`
rec_owner	`pre: rec.owner != rd` `post: result.status == RMI_ERROR_REC`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_bound	`pre: !PaIsTracked(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_gran_state	`pre: GranuleAt(pdev_ptr).state != GRAN_PDEV` `post: result.status == RMI_ERROR_INPUT`
vdev_align	`pre: !AddrIsRmiGranuleAligned(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_bound	`pre: !PaIsTracked(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_gran_state	`pre: GranuleAt(vdev_ptr).state != GRAN_VDEV` `post: result.status == RMI_ERROR_INPUT`
vdev_pdev	`pre: vdev.pdev != pdev_ptr` `post: result.status == RMI_ERROR_DEVICE`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
base_bound	`pre: base != rec.dev_mem_addr` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
top_bound	pre: UInt(top) > UInt(rec.dev_mem_top) post: result.status == RMI_ERROR_INPUT
base_align	pre: !AddrIsRttLevelAligned(base, walk.level) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
top_gran_align	pre: !AddrIsRmiGranuleAligned(top) post: result.status == RMI_ERROR_INPUT
no_progress	pre: UInt(base) == UInt(walk_top_pre) post: (result.status == RMI_ERROR_RTT && result.data.level.level == 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: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
ncoh_pa	pre: (rec.dev_mem_flags.coh == DEV_MEM_NON_COHERENT && !RttEntriesInRangeNonCohDevMem(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre)) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
coh_attr	pre: (rec.dev_mem_flags.coh == DEV_MEM_COHERENT && !RttEntriesInRangeMemAttr(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre, MEMATTR_PASSTHROUGH)) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
coh_pa	pre: (rec.dev_mem_flags.coh == DEV_MEM_COHERENT && !RttEntriesInRangeCohDevMem(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre)) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
linear_map	pre: !RttEntriesInRangeOutputContiguous(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre, rec.dev_mem_pa) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
aux_live	pre: AddrRangeIsAuxLive(base, top, realm_pre) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)

B4.5.63.2.1 Failure condition ordering

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
[rd_bound, rd_state] < [base_align]
[rd_bound, rd_state] < [no_progress]
[rec_bound, rec_gran_state] < [rec_state, rec_owner]
[pdev_bound, pdev_gran_state, vdev_bound, vdev_gran_state] <
     [vdev_pdev]
[base_bound] < [base_align]
[top_gran_align] < [no_progress]
```

B4.5.63.3 Success conditions

ID	Condition
rtte_ripas	`post: RttEntriesInRangeRipas( RttAt(walk.rtt_addr), walk.level, base, walk_top_pre, RIPAS_DEV)`
dev_mem_addr	`post: rec.dev_mem_addr == MinAddress(top, walk_top_pre)`
dev_mem_pa	`post: rec.dev_mem_pa == ToAddress( UInt(pa_pre) + (UInt(walk_top_pre) - UInt(base)))`
out_top	`post: out_top == MinAddress(top, walk_top_pre)`

B4.5.63.4 Footprint

ID	Value
rtte	`RttAt(walk.rtt_addr)`
dev_mem_addr	`rec.dev_mem_addr`
dev_mem_pa	`rec.dev_mem_pa`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.64 RMI_RTT_FOLD command

Destroys a homogeneous primary RTT.

The RMI_RTT_FOLD command may initiate a Stateful RMI Operation.

See also:

- A5.6 *Realm Translation Table*
- A5.6.6 *RTT folding*
- B4.5.56 *RMI_RTT_CREATE command*
- B4.5.60 *RMI_RTT_DESTROY command*

B4.5.64.1 Interface

B4.5.64.1.1 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

B4.5.64.1.2 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

B4.5.64.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.64.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

level_bound	`pre: (!RttLevelIsValid(realm, level)`
	`|| RttLevelIsStarting(realm, level))`
	`post: result.status == RMI_ERROR_INPUT`

ipa_align	`pre: !AddrIsRttLevelAligned(ipa, level - 1)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: UInt(ipa) >= (2 ^ realm.ipa_width)` `post: result.status == RMI_ERROR_INPUT`

rtt_walk	`pre: walk.level < level - 1`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

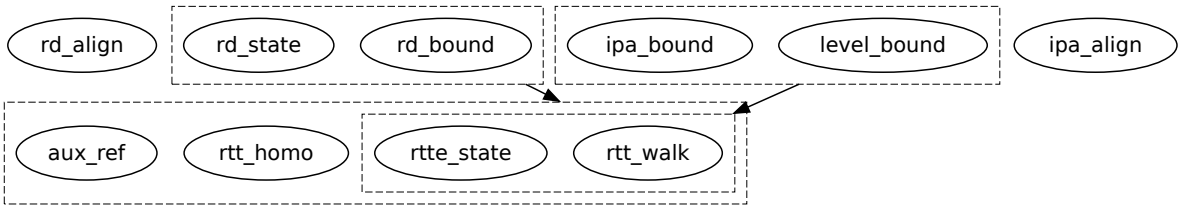
rtte_state	`pre: walk.rtte.state != RTTE_TABLE`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

rtt_homo	`pre: !RttIsHomogeneous(RttAt(walk.rtte.addr))`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == level)`

aux_ref	`pre: AddrIsAuxRef(ipa, realm)`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

B4.5.64.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [rtt_walk, rtte_state, rtt_homo, aux_ref]
[level_bound, ipa_bound] < [rtt_walk, rtte_state]
```



B4.5.64.3 Success conditions

ID	Condition
rtt	`post: rtt == walk.rtte.addr`
result	`post: result.status == RMI_SUCCESS`
rtte_state	`post: walk.rtte.state == fold_pre.state`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rtte_addr	pre: fold_pre.state != RTTE_VOID && fold_pre.state != RTTE_UNMAPPED_NS post: walk.rtte.addr == fold_pre.addr
rtte_attr_prot	pre: fold_pre.state == RTTE_DATA post: (RttMemAttrEqual(walk.rtte, fold_pre, RTT_PROTECTED) && RttS2APEqual(walk.rtte, fold_pre, S2AP_INDIRECT))
rtte_attr_unprot	pre: fold_pre.state == RTTE_MAPPED_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	post: GranuleAt(walk.rtte.addr).state == GRAN_DELEGATED

B4.5.64.4 Footprint

ID	Value
rtt_state	GranuleAt(walk.rtte.addr).state
rtte	RttEntryAt(RttAt(walk.rtt_addr), entry_idx)

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.65 RMI_RTT_INIT_RIPAS command

Set the RIPAS of a target IPA range to RIPAS_RAM, for a Realm in the REALM_NEW state.

The RMI_RTT_INIT_RIPAS command may initiate a Stateful RMI Operation.

See also:

- A5.2.2 *Realm IPA state*
- B4.3.5 *Range RMI operations*
- D1.2.3 *Initialize memory of New Realm flow*

B4.5.65.1 Interface

B4.5.65.1.1 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

B4.5.65.1.2 Context

The RMI_RTT_INIT_RIPAS command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
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	`RttSkipEntriesUnlessVoidOrData(`
		`  ↪ RttAt(walk.rtt_addr),`
		`  walk.level,`
		`  base, top)`
			false	Top IPA of RTTE_VOID entries, starting from entry at which the RTT walk terminated

B4.5.65.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range whose RIPAS was modified

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

IYLLSY If result is not RMI_SUCCESS then the value of out_top is UNKNOWN.

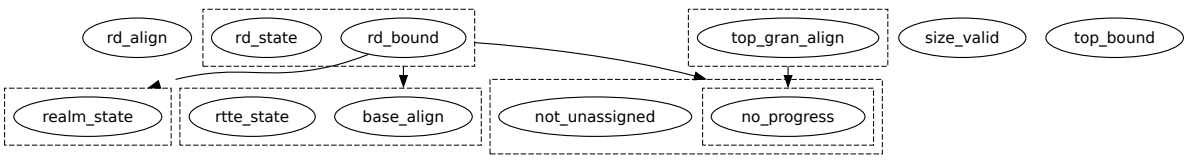
When the out_top output value is valid, it is aligned to the size of the address range described by the RTT entry at the level where the RTT walk terminated.

B4.5.65.2 Failure conditions

ID	Condition
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
size_valid	pre: UInt(top) <= UInt(base) post: result.status == RMI_ERROR_INPUT
top_bound	pre: !AddrIsProtected(ToAddress(UInt(top) - rmm.dynamic.rmi_granule_size), realm_pre) post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm_pre.state != REALM_NEW post: result.status == RMI_ERROR_REALM
base_align	pre: !AddrIsRttLevelAligned(base, walk.level) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
rtte_state	pre: (walk.rtte.state != RTTE_VOID && walk.rtte.state != RTTE_DATA) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
top_gran_align	pre: !AddrIsRmiGranuleAligned(top) post: result.status == RMI_ERROR_INPUT
no_progress	pre: UInt(base) == UInt(walk_top) post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)
not_unassigned	pre: Command encounters RTT entry whose state is neither RTTE_VOID nor RTTE_DATA. post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)

B4.5.65.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[rd_bound, rd_state] < [base_align, rtte_state]
[rd_bound, rd_state] < [no_progress, not_unassigned]
[top_gran_align] < [no_progress]
```



Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.65.3 Success conditions

ID	Condition
rtte_ripas	`post: RttEntriesInRangeRipas( RttAt(walk.rtt_addr), walk.level, base, walk_top, RIPAS_RAM)`
out_top	`post: out_top == walk_top`

B4.5.65.4 Footprint

ID	Value
rtte	`RttAt(walk.rtt_addr)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.66 RMI_RTT_READ_ENTRY command

Reads an entry from a primary RTT.

See also:

- A5.6 *Realm Translation Table*

B4.5.66.1 Interface

B4.5.66.1.1 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

B4.5.66.1.2 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

B4.5.66.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
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 `desc` 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:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- *Arm Architecture Reference Manual for A-Profile architecture* [3]
- A5.6.12 *Memory attributes*

B4.5.66.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
level_bound	`pre: !RttLevelIsValid(realm, level)` `post: result.status == RMI_ERROR_INPUT`
ipa_align	`pre: !AddrIsRttLevelAligned(ipa, level)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: UInt(ipa) >= (2 ^ realm.ipa_width)` `post: result.status == RMI_ERROR_INPUT`

B4.5.66.2.1 Failure condition ordering

The RMI_RTT_READ_ENTRY command does not have any failure condition orderings.

B4.5.66.3 Success conditions

ID	Condition
walk_level	`post: walk_level == walk.level`
state	`post: state == RttEntryStateToRmi(walk.rtte.state)`

state_invalid	`pre: (walk.rtte.state == RTTE_VOID`
	`  || walk.rtte.state == RTTE_UNMAPPED_NS)`
	`post: (rtte.attr_unprot == Zeros{3}()`
	`  && rtte.s2ap_indirect.base_index == S2AP_NO_ACCESS`
	`  && 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 == RTTE_DATA`
	`  || walk.rtte.state == RTTE_NARCH_DEV`
	`  || walk.rtte.state == RTTE_ARCH_DEV`
	`  || walk.rtte.state == RTTE_TABLE)`
	`post: (rtte.attr_unprot == Zeros{3}()`
	`  && rtte.s2ap_indirect.base_index == S2AP_NO_ACCESS`
	`  && rtte.s2ap_indirect.overlay_index == 0`
	`  && rtte.s2ap_direct.read == RMM_FALSE`
	`  && rtte.s2ap_direct.write == RMM_FALSE`
	`  && rtte.addr == walk.rtte.addr)`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
state_unprot	pre: walk.rtte.state == RTTE_MAPPED_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)
state_io	pre: walk.rtte.state == RTTE_NARCH_DEV post: (rtte.attr_unprot == Zeros{3}() && rtte.s2ap_indirect.base_index == S2AP_NO_ACCESS && rtte.s2ap_indirect.overlay_index == 0 && rtte.s2ap_direct.read == RMM_FALSE && rtte.s2ap_direct.write == RMM_FALSE && rtte.addr == walk.rtte.addr)
state_vsmmu	pre: walk.rtte.state == RTTE_ARCH_DEV post: (rtte.attr_unprot == Zeros{3}() && rtte.s2ap_indirect.base_index == S2AP_NO_ACCESS && rtte.s2ap_indirect.overlay_index == 0 && rtte.s2ap_direct.read == RMM_FALSE && rtte.s2ap_direct.write == RMM_FALSE && rtte.addr == walk.rtte.addr)
ripas_prot	pre: (walk.rtte.state == RTTE_VOID || walk.rtte.state == RTTE_DATA) post: ripas == RipasToRmi(walk.rtte.ripas)
ripas_unprot	pre: (walk.rtte.state == RTTE_UNMAPPED_NS || walk.rtte.state == RTTE_MAPPED_NS) post: ripas == RMI_RIPAS_EMPTY

B4.5.66.4 Footprint

The RMI_RTT_READ_ENTRY command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.67 RMI_RTT_SET_RIPAS command

Completes a request made by the Realm to change the RIPAS of a target IPA range.

The RMI_RTT_SET_RIPAS command may initiate a Stateful RMI Operation.

See also:

- A5.4 *RIPAS change*

B4.5.67.1 Interface

B4.5.67.1.1 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

B4.5.67.1.2 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_value`
		`  == RIPAS_RAM) &&`
		`  (rec.ripas_destroyed`
		`  !=`
		`  CHANGE_DESTROYED))`
			true	Top IPA of entries which have associated RIPAS values, starting from entry at which the RTT walk terminated

B4.5.67.1.3 Output values

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range whose RIPAS was modified

I_{ZTWQF} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.67.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
rec_align	`pre: !AddrIsRmiGranuleAligned(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_bound	`pre: !PaIsTracked(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_gran_state	`pre: GranuleAt(rec_ptr).state != GRAN_REC` `post: result.status == RMI_ERROR_INPUT`
rec_state	`pre: rec.state == REC_RUNNING` `post: result.status == RMI_ERROR_REC`
rec_owner	`pre: rec.owner != rd` `post: result.status == RMI_ERROR_REC`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
base_bound	`pre: base != rec.ripas_addr` `post: result.status == RMI_ERROR_INPUT`
top_bound	`pre: UInt(top) > UInt(rec.ripas_top)` `post: result.status == RMI_ERROR_INPUT`

base_align	`pre: (!AddrIsRttLevelAligned(base, walk.level)`
	`  && ripas_pre != rec.ripas_value)`
	`post: (result.status == RMI_ERROR_RTT`
	`  && result.data.level.level == walk.level)`

top_gran_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`

no_progress	`pre: (UInt(base) == UInt(walk_top_pre)`
	`  && ripas_pre != rec.ripas_value)`
	`post: (result.status == RMI_ERROR_RTT`
	`  && result.data.level.level == walk.level)`

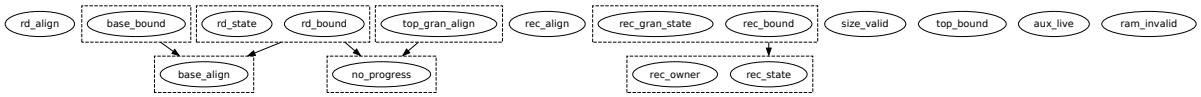
aux_live	`pre: AddrRangeIsAuxLive(base, top, realm_pre)`
	`post: (result.status == RMI_ERROR_RTT`
	`  && result.data.level.level == walk.level)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
ram_invalid	pre: ripas == RIPAS_RAM and command encounters RTT entry whose state is neither RTTE_DATA nor RTTE_VOID. post: (result.status == RMI_ERROR_RTT && result.data.level.level == walk.level)

B4.5.67.2.1 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]
```



B4.5.67.3 Success conditions

ID	Condition
rtte_ripas	post: RttEntriesInRangeRipas(RttAt(walk.rtt_addr), walk.level, base, walk_top_pre, rec.ripas_value)
ripas_addr	post: rec.ripas_addr == MinAddress(top, walk_top_pre)
out_top	post: out_top == MinAddress(top, walk_top_pre)

B4.5.67.4 Footprint

ID	Value
rtte	RttAt(walk.rtt_addr)
ripas_addr	rec.ripas_addr

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.68 RMI_RTT_SET_S2AP command

Completes a request made by the Realm to change the S2AP of a target IPA range.

The RMI_RTT_SET_S2AP command may initiate a Stateful RMI Operation.

See also:

- A10.3.2.3 *Stage 2 Access Permissions change within a multi-Plane Realm*
- B4.3.5 *Range RMI operations*

B4.5.68.1 Interface

B4.5.68.1.1 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

B4.5.68.1.2 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

B4.5.68.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
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

I_{BWKVD} 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.68.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
rec_align	`pre: !AddrIsRmiGranuleAligned(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_bound	`pre: !PaIsTracked(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_gran_state	`pre: GranuleAt(rec_ptr).state != GRAN_REC` `post: result.status == RMI_ERROR_INPUT`
rec_state	`pre: rec.state == REC_RUNNING` `post: result.status == RMI_ERROR_REC`
rec_owner	`pre: rec.owner != rd` `post: result.status == RMI_ERROR_REC`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
base_bound	`pre: base != rec.s2ap_addr` `post: result.status == RMI_ERROR_INPUT`
top_bound	`pre: UInt(top) > UInt(rec.s2ap_top)` `post: result.status == RMI_ERROR_INPUT`
top_gran_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == 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: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == not_aligned.walk.level)`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition

base_align_aux	`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: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == not_aligned.walk.level)`

B4.5.68.2.1 Failure condition ordering

The RMI_RTT_SET_S2AP command does not have any failure condition orderings.

B4.5.68.3 Success conditions

ID	Condition
s2ap_addr	`post: rec.s2ap_addr == out_top`

B4.5.68.4 Footprint

ID	Value
s2ap_addr	`rec.s2ap_addr`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.69 RMI_RTT_UNPROT_MAP command

Create mappings to conventional memory within a target Unprotected IPA range.

The RMI_RTT_UNPROT_MAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.11 *Create mappings from Unprotected IPA space*
- B4.5.70 *RMI_RTT_UNPROT_UNMAP command*

B4.5.69.1 Interface

B4.5.69.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001FB`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttUnprotMapFlags	Flags
oaddr	X5	63:0	RmiAddrSetDesc	Output address set descriptor. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this describes a contiguous PA range which will be mapped into the target IPA range. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the PA of a Granule that holds an RMI Address Range List. This describes a list of PA regions which will be mapped into the target IPA range.

DRAFT

B4.5.69.1.2 Context

The RMI_RTT_UNPROT_MAP command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes
progress	UInt64	`UInt(out_top) - UInt(base)`	false	Size of IPA range which has been mapped

walk	RmmRttWalkResult	`RttWalk(`
		`  realm, base,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	Result of RTT walk from base of the target IPA range

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.69.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been mapped

I_{HLVMB} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.69.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
base_align	`pre: !AddrIsRmiGranuleAligned(base)` `post: result.status == RMI_ERROR_INPUT`
top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: AddrIsProtected(base, realm)` `post: result.status == RMI_ERROR_INPUT`

oaddr_type	`pre: (flags.oaddr_type != RMI_ADDR_TYPE_SINGLE`
	`  && flags.oaddr_type != RMI_ADDR_TYPE_LIST)`
	`post: result.status == RMI_ERROR_INPUT`

oaddr_align	`pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST`
	`  && !AddrIsAligned(oaddr.data.list_addr.addr, 8))`
	`post: result.status == RMI_ERROR_INPUT`

oaddr_list_pas	`pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST`
	`  && !NonSecureAccessPermitted(`
	`  oaddr.data.list_addr.addr))`
	`post: result.status == RMI_ERROR_INPUT`

rtte_state	`pre: walk.rtte.state != RTTE_UNMAPPED_NS`
	`post: (result.status == RMI_ERROR_RTT`
	`  && result.data.level.level == walk.level)`

rtte_size	`pre: (walk.rtte.state == RTTE_VOID`
	`  && RttLevelSize(walk.level) > size)`
	`post: (result.status == RMI_ERROR_RTT`
	`  && result.data.level.level == walk.level)`

B4.5.69.2.1 Failure condition ordering

The RMI_RTT_UNPROT_MAP command does not have any failure condition orderings.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.69.3 Success conditions

ID	Condition

state	`post: RttTreeRangeAllState(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base, out_top, RTTE_MAPPED_NS)`

addr_contig	`pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE`
	`post: RttTreeRangeAllOaddrContig(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base,`
	`RmiAddrRangeDescDecode(oaddr.data.single).base,`
	`progress)`

addr_list	`pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST`
	`post: RttTreeRangeAllOaddrList(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base,`
	`oaddr.data.list_addr.addr,`
	`progress)`

memattr	`post: RttTreeRangeAllMemAttr(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base, out_top, flags.memattr)`

s2ap	`post: RttTreeRangeAllS2AP(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base, out_top, flags.s2ap)`

result	`post: result.status == RMI_SUCCESS`

B4.5.69.4 Footprint

The RMI_RTT_UNPROT_MAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.70 RMI_RTT_UNPROT_UNMAP command

Removes mappings to conventional memory within a target Unprotected IPA range.

The RMI_RTT_UNPROT_UNMAP command may initiate a Stateful RMI Operation.

See also:

- A5.3.12 *Remove mappings from Unprotected IPA space*
- B4.5.69 *RMI_RTT_UNPROT_MAP command*

B4.5.70.1 Interface

B4.5.70.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001FC`
rd	X1	63:0	Address	PA of the RD for the target Realm
base	X2	63:0	Address	Base of the target IPA range
top	X3	63:0	Address	Top of the target IPA range
flags	X4	63:0	RmiRttUnmapFlags	Flags
oaddr	X5	63:0	RmiAddrSetDesc	Output address set descriptor. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this SBZ. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the PA of a Granule that holds an RMI Address Range List. This describes a list of PA regions which have been unmapped from the target IPA range.

DRAFT

B4.5.70.1.2 Context

The RMI_RTT_UNPROT_UNMAP command operates on the following context.

Name	Type	Value	Before	Description
realm_pre	RmmRealm	`RealmAt(rd)`	true	Realm
realm	RmmRealm	`RealmAt(rd)`	false	Realm
size	UInt64	`UInt(top) - UInt(base)`	false	Size of target IPA range in bytes
progress	UInt64	`UInt(out_top) - UInt(base)`	false	Size of IPA range which has been unmapped
walk_pre	RmmRttWalkResult	`RttWalk( realm, base, RMM_RTT_PAGE_LEVEL, RMM_RTT_TREE_PRIMARY)`	true	Result of RTT walk from base of the target IPA range

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Value	Before	Description

walk	RmmRttWalkResult	`RttWalk(`
		`realm, base,`
		`RMM_RTT_PAGE_LEVEL,`
		`RMM_RTT_TREE_PRIMARY)`
			false	Result of RTT walk from base of the target IPA range

B4.5.70.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
out_top	X1	63:0	Address	Top IPA of range which has been unmapped
out_range	X2	63:0	RmiAddrRangeDesc	Output address range. If flags.oaddr_type == RMI_ADDR_TYPE_SINGLE then this describes a contiguous PA range which has been unmapped from the target IPA range. If flags.oaddr_type != RMI_ADDR_TYPE_SINGLE then this value is zero.
out_count	X3	63:0	UInt64	Number of entries in output address list. If flags.oaddr_type == RMI_ADDR_TYPE_LIST then this is the number of entries which have been written to the output address list. If flags.oaddr_type != RMI_ADDR_TYPE_LIST then this value is zero.

DRAFT

I_{QXLJL} If `result` is not RMI_SUCCESS then the value of `out_top` is UNKNOWN.

B4.5.70.2 Failure conditions

ID	Condition
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
base_align	`pre: !AddrIsRmiGranuleAligned(base)` `post: result.status == RMI_ERROR_INPUT`
top_align	`pre: !AddrIsRmiGranuleAligned(top)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
size_valid	`pre: UInt(top) <= UInt(base)` `post: result.status == RMI_ERROR_INPUT`
ipa_bound	`pre: AddrIsProtected(base, realm)` `post: result.status == RMI_ERROR_INPUT`

oaddr_align	`pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST`
	`&& !AddrIsAligned(oaddr.data.list_addr.addr, 8))`
	`post: result.status == RMI_ERROR_INPUT`

oaddr_list_pas	`pre: (flags.oaddr_type == RMI_ADDR_TYPE_LIST`
	`&& !NonSecureAccessPermitted(`
	`oaddr.data.list_addr.addr))`
	`post: result.status == RMI_ERROR_INPUT`

rtte_state	`pre: walk.rtte.state != RTTE_MAPPED_NS`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

rtte_size	`pre: (walk.rtte.state == RTTE_MAPPED_NS`
	`&& RttLevelSize(walk.level) > size)`
	`post: (result.status == RMI_ERROR_RTT`
	`&& result.data.level.level == walk.level)`

B4.5.70.2.1 Failure condition ordering

The RMI_RTT_UNPROT_UNMAP command does not have any failure condition orderings.

B4.5.70.3 Success conditions

ID	Condition

state	`post: RttTreeRangeAllState(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base, out_top, RTTE_UNMAPPED_NS)`

addr_single	`pre: flags.oaddr_type == RMI_ADDR_TYPE_SINGLE` `post: out_range.data.addr == walk_pre.rtte.addr`

addr_list	`pre: flags.oaddr_type == RMI_ADDR_TYPE_LIST`
	`post: RttTreeRangeAllOaddrList(`
	`realm, RMM_RTT_TREE_PRIMARY,`
	`base,`
	`oaddr.data.list_addr.addr,`
	`progress)`

result	`post: result.status == RMI_SUCCESS`

B4.5.70.4 Footprint

The RMI_RTT_UNPROT_UNMAP command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.71 RMI_VDEV_ABORT command

Abort device communication associated with a VDEV.

See also:

- Chapter A9 *Realm device assignment*

B4.5.71.1 Interface

B4.5.71.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000185`
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

B4.5.71.1.2 Context

The RMI_VDEV_ABORT command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV

DRAFT

B4.5.71.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.71.2 Failure conditions

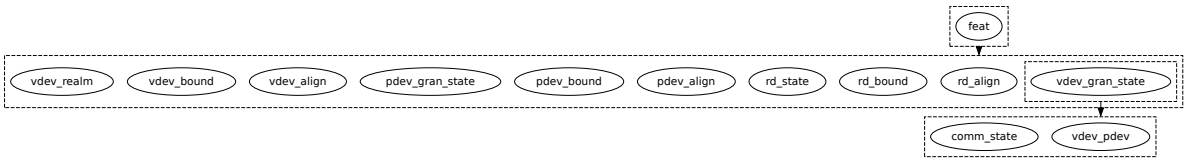
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsTracked(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev.realm != rd post: result.status == RMI_ERROR_INPUT
vdev_pdev	pre: vdev.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE
comm_state	pre: vdev.comm_state == DEV_COMM_IDLE post: result.status == RMI_ERROR_DEVICE

B4.5.71.2.1 Failure condition ordering

```
[feat] < [rd_align, rd_bound, rd_state, pdev_align, pdev_bound,
     pdev_gran_state, vdev_align, vdev_bound, vdev_gran_state,
     vdev_realm]
[vdev_gran_state] < [vdev_pdev, comm_state]
```



B4.5.71.3 Success conditions

ID	Condition
state	post: vdev.vdev_state == VDEV_ERROR
comm_state	post: vdev.comm_state == DEV_COMM_IDLE

B4.5.71.4 Footprint

ID	Value
state	vdev.vdev_state
comm_state	vdev.comm_state

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.72 RMI_VDEV_COMMUNICATE command

Perform device communication associated with a VDEV.

See also:

- Chapter A9 *Realm device assignment*

B4.5.72.1 Interface

B4.5.72.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000186`
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
data_ptr	X4	63:0	Address	PA of the communication data structure

UMVSFY 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.

B4.5.72.1.2 Context

The RMI_VDEV_COMMUNICATE command operates on the following context.

Name	Type	Value	Before	Description
rmm	RmmGlobal	`Rmm()`	false	RMM global state
realm	RmmRealm	`RealmAt(rd)`	false	Realm
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev_pre	RmmVdev	`VdevAt(vdev_ptr)`	true	VDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV
data	RmiDevCommData	`RmiDevCommDataAt(data_ptr)`	false	Device communication object

B4.5.72.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.72.2 Failure conditions

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

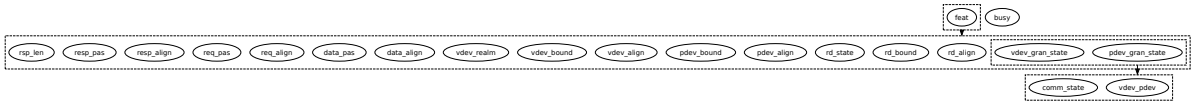
ID	Condition
feat	pre: Rmm().static.feat_da != FEATURE_TRUE post: result.status == RMI_ERROR_NOT_SUPPORTED
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsTracked(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev.realm != rd post: result.status == RMI_ERROR_INPUT
data_align	pre: !AddrIsRmiGranuleAligned(data_ptr) post: result.status == RMI_ERROR_INPUT
data_pas	pre: !NonSecureAccessPermitted(data_ptr) post: result.status == RMI_ERROR_INPUT
req_align	pre: !AddrIsRmiGranuleAligned(data.enter.req_addr) post: result.status == RMI_ERROR_INPUT
req_pas	pre: !NonSecureAccessPermitted(data.enter.req_addr) post: result.status == RMI_ERROR_INPUT
resp_align	pre: !AddrIsRmiGranuleAligned(data.enter.rsp_addr) post: result.status == RMI_ERROR_INPUT
resp_pas	pre: !NonSecureAccessPermitted(data.enter.rsp_addr) post: result.status == RMI_ERROR_INPUT
rsp_len	pre: data.enter.rsp_len > rmm.dynamic.rmi_granule_size post: result.status == RMI_ERROR_INPUT
vdev_pdev	pre: vdev.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE
comm_state	pre: vdev.comm_state == DEV_COMM_IDLE post: result.status == RMI_ERROR_DEVICE
busy	pre: PdevIsBusy(pdev) post: result.status == RMI_BUSY

B4.5.72.2.1 Failure condition ordering

```
[feat] < [rd_align, rd_bound, rd_state, pdev_align, pdev_bound,
     pdev_gran_state, vdev_align, vdev_bound, vdev_gran_state,
     vdev_realm, data_align, data_pas, req_align, req_pas, resp_align,
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
     resp_pas, rsp_len]
[pdev_gran_state, vdev_gran_state] < [vdev_pdev, comm_state]
```



B4.5.72.3 Success conditions

ID	Condition
comm_state	`post: vdev.comm_state == DeviceCommunicate(vdev, data)`
error	`pre: DeviceCommunicate(vdev, data) == DEV_COMM_ERROR` `post: vdev.vdev_state == VDEV_ERROR`

locked_unlocked	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.vdev_state == VDEV_LOCKED`
	`&& vdev.op == VDEV_OP_UNLOCK)`
	`post: vdev.vdev_state == VDEV_UNLOCKED`

started_unlocked	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.vdev_state == VDEV_STARTED`
	`&& vdev.op == VDEV_OP_UNLOCK`
	`&& rmm.static.feat_vdev_krou == FEATURE_FALSE)`
	`post: vdev.vdev_state == VDEV_UNLOCKED`

started_key_refresh	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.vdev_state == VDEV_STARTED`
	`&& vdev.op == VDEV_OP_UNLOCK`
	`&& rmm.static.feat_vdev_krou == FEATURE_TRUE)`
	`post: vdev.vdev_state == VDEV_KEY_REFRESH`

key_refresh	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.op == VDEV_OP_KEY_REFRESH)`
	`post: vdev.vdev_state == VDEV_KEY_PURGE`

key_purge	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.op == VDEV_OP_KEY_PURGE)`
	`post: vdev.vdev_state == VDEV_UNLOCKED`

lock_state	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.op == VDEV_OP_LOCK)`
	`post: vdev.vdev_state == VDEV_LOCKED`

lock_nonce	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.op == VDEV_OP_LOCK)`
	`post: vdev.attest_info.lock_nonce == VdevGenerateNonce(vdev_pre)`

start_state	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.op == VDEV_OP_START)`
	`post: vdev.vdev_state == VDEV_STARTED`

meas_nonce	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.op == VDEV_OP_GET_MEAS)`
	`post: vdev.attest_info.meas_nonce == VdevGenerateNonce(vdev_pre)`

intf_count	`pre: (DeviceCommunicate(vdev, data) == DEV_COMM_IDLE`
	`&& vdev.op == VDEV_OP_GET_REPORT)`
	`post: vdev.attest_info.report_nonce == VdevGenerateNonce(vdev_pre)`

op	`pre: DeviceCommunicate(vdev, data) == DEV_COMM_IDLE` `post: vdev.op == VDEV_OP_NONE`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
complete	pre: DeviceCommunicate(vdev, data) == DEV_COMM_IDLE post: RmiDevCommComplete(data.exit.flags)
incomplete	pre: DeviceCommunicate(vdev, data) != DEV_COMM_IDLE post: !RmiDevCommComplete(data.exit.flags)

B4.5.72.4 Footprint

ID	Value
state	vdev.vdev_state
op	vdev.op
comm_state	vdev.comm_state
lock_nonce	vdev.attest_info.lock_nonce
meas_nonce	vdev.attest_info.meas_nonce
report_nonce	vdev.attest_info.report_nonce

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.73 RMI_VDEV_COMPLETE command

Completes a pending VDEV request.

See also:

- A4.3.12 *REC exit due to VDEV request*
- A9.4.4 *Mapping from virtual device ID to VDEV object*

B4.5.73.1 Interface

B4.5.73.1.1 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

B4.5.73.1.2 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

DRAFT

B4.5.73.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.73.2 Failure conditions

ID	Condition
rec_align	`pre: !AddrIsRmiGranuleAligned(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
rec_bound	`pre: !PaIsTracked(rec_ptr)` `post: result.status == RMI_ERROR_INPUT`
recv_state	`pre: GranuleAt(rec_ptr).state != GRAN_REC` `post: result.status == RMI_ERROR_INPUT`
vdev_align	`pre: !AddrIsRmiGranuleAligned(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_bound	`pre: !PaIsTracked(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
vdev_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
pending	pre: rec.pending != REC_PENDING_VDEV_REQUEST post: result.status == RMI_ERROR_INPUT
owner	pre: rec.owner != vdev.realm post: result.status == RMI_ERROR_INPUT
vdev_id	pre: rec.vdev_id_1 != vdev.vdev_id post: result.status == RMI_ERROR_INPUT
comm_state	pre: vdev.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_DEVICE

B4.5.73.2.1 Failure condition ordering

The RMI_VDEV_COMPLETE command does not have any failure condition orderings.

B4.5.73.3 Success conditions

ID	Condition
pending	post: rec.pending == REC_PENDING_VDEV_COMPLETE
vdev_pa	post: rec.vdev_pa_1 == vdev_ptr

B4.5.73.4 Footprint

ID	Value
pending	rec.pending
vdev_pa	rec.vdev_pa_1

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.74 RMI_VDEV_CREATE command

Create a VDEV.

The RMI_VDEV_CREATE command may initiate a Stateful RMI Operation.

The RMI_VDEV_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- Chapter A9 *Realm device assignment*
- A9.4.2 *Virtual device invariants*
- B4.3.4 *Object creation and destruction*

B4.5.74.1 Interface

B4.5.74.1.1 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

B4.5.74.1.2 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

stream_result	RmmPdevStreamResult	`PdevStreamFromType(`
		`  pdev,`
		`  PDEV_STREAM_NCOH)`
			false	Result of looking up PDEV stream

psmmu	RmmPsmmu	`PsmmuFromPdev(pdev)`	false	PSMMU
st_walk	RmmPsmmuStWalkResul	`PsmmuStWalk(` `  psmmu, VdevSid(vdev))`	false	Result of PSMMU Stream Table walk

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.74.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.74.2 Failure conditions

ID	Condition
feat	pre: Rmm().static.feat_da != FEATURE_TRUE post: result.status == RMI_ERROR_NOT_SUPPORTED
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
pdev_state	pre: pdev.state != PDEV_READY post: result.status == RMI_ERROR_DEVICE

pdev_category	pre: !pdev.category IN {
	PDEV_ENDPOINT_ACCEL_OFF_CHIP,
	PDEV_ENDPOINT_ACCEL_ON_CHIP
	}
	post: result.status == RMI_ERROR_DEVICE

pdev_streams	pre: PdevStreamsForVdev(pdev) post: result.status == RMI_ERROR_DEVICE
pdev_num_vdevs	pre: pdev.num_vdevs == pdev.max_num_vdevs post: result.status == RMI_ERROR_DEVICE
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsDelegableConventionalFine(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_DELEGATED post: result.status == RMI_ERROR_INPUT
params_align	pre: !AddrIsRmiGranuleAligned(params_ptr) post: result.status == RMI_ERROR_INPUT
params_pas	pre: !NonSecureAccessPermitted(params_ptr) post: result.status == RMI_ERROR_INPUT
params_valid	pre: !RmiVdevParamsIsValid(params_ptr) post: result.status == RMI_ERROR_INPUT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
addr_range_vali d	pre: !RmiAddrRangesValid8(params.addr_range, params.num_addr_range) post: result.status == RMI_ERROR_INPUT
da_en	pre: realm.feat_da != FEATURE_TRUE post: result.status == RMI_ERROR_REALM
vdev_id_free	pre: !VdevIdIsFree(realm, params.vdev_id) post: result.status == RMI_ERROR_INPUT
tdi_id_free	pre: !TdiIdIsFree(params.tdi_id, pdev.routing_id) post: result.status == RMI_ERROR_INPUT
stream_exists	pre: stream_result.valid != RMM_TRUE post: result.status == RMI_ERROR_INPUT
tdi_id_bound	pre: (UInt(params.tdi_id) < UInt(pdev.rid_base) || UInt(params.tdi_id) >= UInt(pdev.rid_top)) post: result.status == RMI_ERROR_INPUT
psmmu_st_l2	pre: (st_walk.level != 2 || st_walk.ste.state != PSMMU_ST_ENTRY_INVALID) post: result.status == RMI_ERROR_INPUT
vsmmu_align	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && !AddrIsRmiGranuleAligned(params.vsmmu_addr)) post: result.status == RMI_ERROR_INPUT
vsmmu_bound	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && !PaIsTracked(params.vsmmu_addr)) post: result.status == RMI_ERROR_INPUT
vsmmu_state	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && GranuleAt(params.vsmmu_addr).state != GRAN_VSMMU) post: result.status == RMI_ERROR_INPUT
vsid_free	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && !VsidIsFree(VsmmuAt(params.vsmmu_addr), params.vsid)) post: result.status == RMI_ERROR_INPUT
vsmmu_compat	pre: (params.flags.VSMMU == RMI_FEATURE_TRUE && !PdevVsmmuIsCompatible(pdev, VsmmuAt(params.vsmmu_addr))) post: result.status == RMI_ERROR_INPUT

B4.5.74.2.1 Failure condition ordering

```
[feat] < [rd_align, rd_bound, pdev_bound, pdev_gran_state, vdev_align,
     vdev_bound, vdev_gran_state, params_align, params_pas,
     params_valid, vsmmu_align, vsmmu_bound, vsmmu_state, vsid_free]
[feat] < [pdev_gran_state]
[feat] < [rd_state]
[pdev_gran_state, vsmmu_state] < [vsmmu_compat]
[pdev_gran_state] < [pdev_category, pdev_state, pdev_num_vdevs,
     pdev_streams]
[rd_state] < [da_en]
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.74.3 Success conditions

ID	Condition
pdev_num_vdevs	`post: pdev.num_vdevs == num_vdevs_pre + 1`
gran_state	`post: GranuleAt(vdev_ptr).state == GRAN_VDEV`
vdev_id	`post: vdev.vdev_id == params.vdev_id`
tdi_id	`post: vdev.tdi_id == params.tdi_id`
pdev	`post: vdev.pdev == pdev_ptr`
realm	`post: vdev.realm == rd`
vdev_state	`post: vdev.vdev_state == VDEV_NEW`
dma_state	`post: vdev.dma_state == VDEV_DMA_DISABLED`
op	`post: vdev.op == VDEV_OP_UNLOCK`
comm_state	`post: vdev.comm_state == DEV_COMM_PENDING`
tdi_id_used	`post: !TdiIdIsFree(params.tdi_id, pdev.routing_id)`
vsmmu	`post: 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	`post: realm.num_vdevs == realm_pre.num_vdevs + 1`
lock_nonce	`post: vdev.attest_info.lock_nonce == 0`
meas_nonce	`post: vdev.attest_info.meas_nonce == 0`
report_nonce	`post: vdev.attest_info.report_nonce == 0`
num_addr_range	`post: vdev.num_addr_range == params.num_addr_range`

addr_range	`post: RmiAddrRangesEqual8(`
	`vdev.addr_range,`
	`params.addr_range,`
	`params.num_addr_range)`

ste_state	`post: st_walk.ste.state == PSMMU_ST_ENTRY_VALID`

B4.5.74.4 Footprint

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Value
state	`GranuleAt(vdev_ptr).state`
pdev_num_vdevs	`pdev.num_vdevs`
realm_num_vdevs	`realm.num_vdevs`
ste_state	`st_walk.ste.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.75 RMI_VDEV_DESTROY command

Destroy a VDEV.

The RMI_VDEV_DESTROY command may initiate a Stateful RMI Operation.

The RMI_VDEV_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- Chapter A9 *Realm device assignment*
- B4.3.4 *Object creation and destruction*

B4.5.75.1 Interface

B4.5.75.1.1 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

B4.5.75.1.2 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
psmmu	RmmPsmmu	`PsmmuFromPdev(pdev)`	false	PSMMU
st_walk	RmmPsmmuStWalkResul	`PsmmuStWalk(psmmu, VdevSid(vdev_pre))`	false	Result of PSMMU Stream Table walk

B4.5.75.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.75.2 Failure conditions

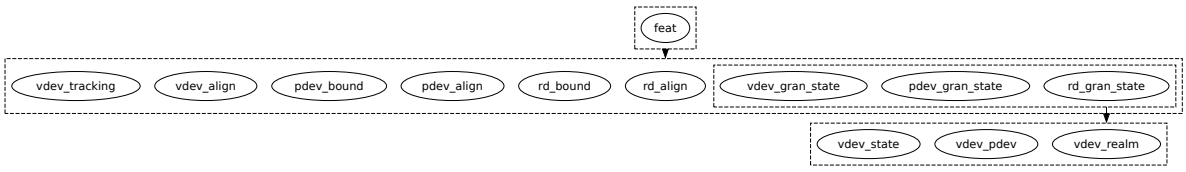
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
feat	pre: Rmm().static.feat_da != FEATURE_TRUE post: result.status == RMI_ERROR_NOT_SUPPORTED
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_gran_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_tracking	pre: !PaIsTrackedFine(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev_pre.realm != rd post: result.status == RMI_ERROR_DEVICE
vdev_pdev	pre: vdev_pre.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE
vdev_state	pre: (vdev_pre.vdev_state != VDEV_NEW && vdev_pre.vdev_state != VDEV_UNLOCKED) post: result.status == RMI_ERROR_DEVICE

B4.5.75.2.1 Failure condition ordering

```
[feat] < [rd_align, rd_bound, rd_gran_state, pdev_align, pdev_bound,
     pdev_gran_state, vdev_align, vdev_tracking, vdev_gran_state]
[rd_gran_state, pdev_gran_state, vdev_gran_state] < [vdev_realm,
     vdev_pdev, vdev_state]
```



B4.5.75.3 Success conditions

ID	Condition
gran_state	post: GranuleAt(vdev_ptr).state == GRAN_DELEGATED
vdev_id_free	post: VdevIdIsFree(realm, vdev_pre.vdev_id)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
tdi_id_free	`post: TdiIdIsFree(vdev_pre.tdi_id, pdev_pre.routing_id)`
realm_num_vdevs	`post: realm.num_vdevs == realm_pre.num_vdevs - 1`
pdev_num_vdevs	`post: pdev.num_vdevs == pdev_pre.num_vdevs - 1`
vsid_free	`pre: vdev_pre.vsmmu == FEATURE_TRUE` `post: VsidIsFree( VsmmuAt(vdev_pre.vsmmu_addr), vdev_pre.vsid)`
ste_state	`post: st_walk.ste.state == PSMMU_ST_ENTRY_INVALID`

B4.5.75.4 Footprint

ID	Value
state	`GranuleAt(vdev_ptr).state`
realm_num_vdevs	`realm.num_vdevs`
pdev_num_vdevs	`pdev.num_vdevs`
ste_state	`st_walk.ste.state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.76 RMI_VDEV_GET_INTERFACE_REPORT command

Get VDEV interface report.

See also:

- A9.6.1 *Realm retrieval of device attestation evidence*

B4.5.76.1 Interface

B4.5.76.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001D0`
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

B4.5.76.1.2 Context

The RMI_VDEV_GET_INTERFACE_REPORT command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV

DRAFT

B4.5.76.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.76.2 Failure conditions

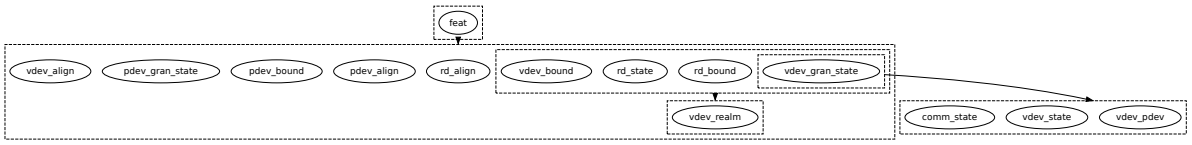
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsTracked(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev.realm != rd post: result.status == RMI_ERROR_INPUT
vdev_pdev	pre: vdev.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE
vdev_state	pre: (vdev.vdev_state != VDEV_LOCKED && vdev.vdev_state != VDEV_STARTED) post: result.status == RMI_ERROR_DEVICE
comm_state	pre: vdev.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_DEVICE

B4.5.76.2.1 Failure condition ordering

```
[rd_bound, rd_state, vdev_bound, vdev_gran_state] < [vdev_realm]
[feat] < [rd_align, rd_bound, rd_state, pdev_align, pdev_bound,
     pdev_gran_state, vdev_align, vdev_bound, vdev_gran_state,
     vdev_realm]
[vdev_gran_state] < [vdev_pdev, vdev_state, comm_state]
```



B4.5.76.3 Success conditions

ID	Condition
op	post: vdev.op == VDEV_OP_GET_REPORT
comm_state	post: vdev.comm_state == DEV_COMM_PENDING

B4.5.76.4 Footprint

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Value
op	`vdev.op`
comm_state	`vdev.comm_state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.77 RMI_VDEV_GET_MEASUREMENTS command

Get VDEV measurements.

See also:

- A9.6.1 *Realm retrieval of device attestation evidence*

B4.5.77.1 Interface

B4.5.77.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001D1`
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

R_{TJPLJ} If pdev.spdm == true and the device supports signed measurement, then:

- The final SPDM GET_MEASUREMENTS request issued by the RMM in reponse to this command includes a request for a signature.
- The RMM first requests the Host to cache the request, and then requests the Host to cache the response. As a result, Host concatenates the request and the response within its cache. The cached data in turn includes any opaque data, and the signature.

B4.5.77.1.2 Context

The RMI_VDEV_GET_MEASUREMENTS command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV
params	RmiVdevMeasureParams	`RmiVdevMeasureParamsAt(params_ptr)`	false	Measurement parameters

B4.5.77.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.77.2 Failure conditions

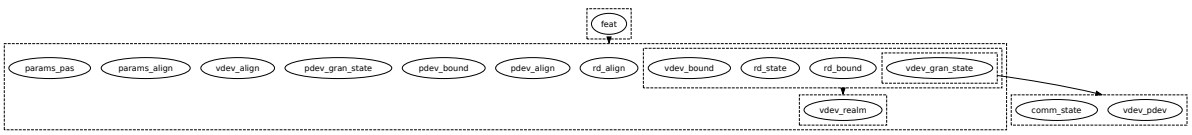
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`
pdev_align	`pre: !AddrIsRmiGranuleAligned(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_bound	`pre: !PaIsTracked(pdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
pdev_gran_state	`pre: GranuleAt(pdev_ptr).state != GRAN_PDEV` `post: result.status == RMI_ERROR_INPUT`
vdev_align	`pre: !AddrIsRmiGranuleAligned(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_bound	`pre: !PaIsTracked(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_gran_state	`pre: GranuleAt(vdev_ptr).state != GRAN_VDEV` `post: result.status == RMI_ERROR_INPUT`
vdev_realm	`pre: vdev.realm != rd` `post: result.status == RMI_ERROR_INPUT`
vdev_pdev	`pre: vdev.pdev != pdev_ptr` `post: result.status == RMI_ERROR_DEVICE`
comm_state	`pre: vdev.comm_state != DEV_COMM_IDLE` `post: result.status == RMI_ERROR_DEVICE`
params_align	`pre: !AddrIsRmiGranuleAligned(params_ptr)` `post: result.status == RMI_ERROR_INPUT`
params_pas	`pre: !NonSecureAccessPermitted(params_ptr)` `post: result.status == RMI_ERROR_INPUT`

B4.5.77.2.1 Failure condition ordering

```
[rd_bound, rd_state, vdev_bound, vdev_gran_state] < [vdev_realm]
[feat] < [rd_align, rd_bound, rd_state, pdev_align, pdev_bound,
     pdev_gran_state, vdev_align, vdev_bound, vdev_gran_state,
     vdev_realm, params_align, params_pas]
[vdev_gran_state] < [vdev_pdev, comm_state]
```



B4.5.77.3 Success conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
op	`post: vdev.op == VDEV_OP_GET_MEAS`
comm_state	`post: vdev.comm_state == DEV_COMM_PENDING`

B4.5.77.4 Footprint

ID	Value
op	`vdev.op`
comm_state	`vdev.comm_state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.78 RMI_VDEV_GET_STATE command

Get state of a VDEV.

See also:

- Chapter A9 *Realm device assignment*

B4.5.78.1 Interface

B4.5.78.1.1 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

B4.5.78.1.2 Context

The RMI_VDEV_GET_STATE command operates on the following context.

Name	Type	Value	Before	Description
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV

B4.5.78.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
state	X1	7:0	RmiVdevState	VDEV state

The following unused bits of RMI_VDEV_GET_STATE output values MBZ: X1[63:8].

B4.5.78.2 Failure conditions

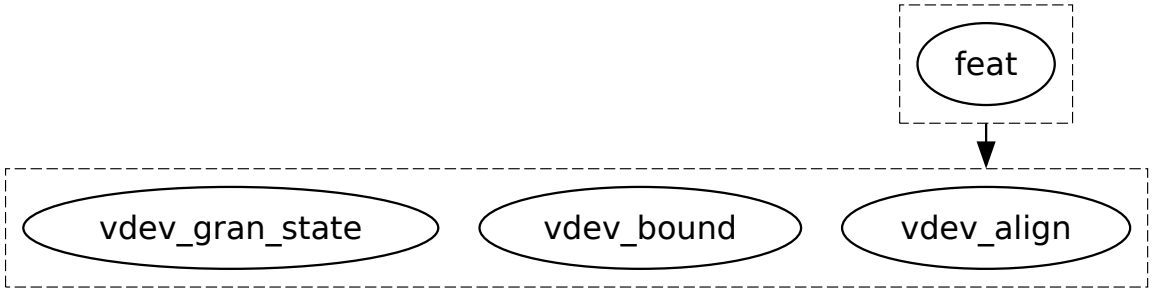
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
vdev_align	`pre: !AddrIsRmiGranuleAligned(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_bound	`pre: !PaIsTracked(vdev_ptr)` `post: result.status == RMI_ERROR_INPUT`
vdev_gran_state	`pre: GranuleAt(vdev_ptr).state != GRAN_VDEV` `post: result.status == RMI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.78.2.1 Failure condition ordering

```
[feat] < [vdev_align, vdev_bound, vdev_gran_state]
```



B4.5.78.3 Success conditions

ID	Condition
state	`post: Equal(state, vdev.vdev_state)`

B4.5.78.4 Footprint

The RMI_VDEV_GET_STATE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.79 RMI_VDEV_LOCK command

Lock VDEV.

See also:

- A9.4.3 *Virtual device lifecycle*

B4.5.79.1 Interface

B4.5.79.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001D2`
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

B4.5.79.1.2 Context

The RMI_VDEV_LOCK command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV

B4.5.79.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.79.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

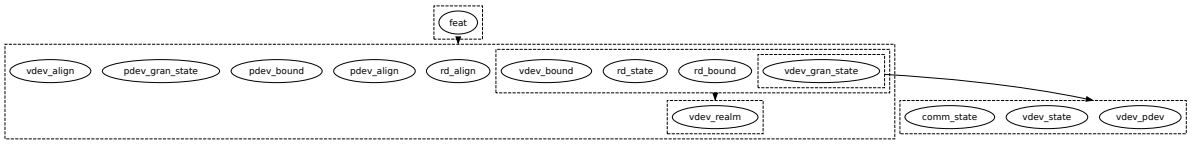
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsTracked(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev.realm != rd post: result.status == RMI_ERROR_INPUT
vdev_pdev	pre: vdev.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE
vdev_state	pre: vdev.vdev_state != VDEV_UNLOCKED post: result.status == RMI_ERROR_DEVICE
comm_state	pre: vdev.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_DEVICE

B4.5.79.2.1 Failure condition ordering

```
[rd_bound, rd_state, vdev_bound, vdev_gran_state] < [vdev_realm]
[feat] < [rd_align, rd_bound, rd_state, pdev_align, pdev_bound,
     pdev_gran_state, vdev_align, vdev_bound, vdev_gran_state,
     vdev_realm]
[vdev_gran_state] < [vdev_pdev, vdev_state, comm_state]
```



B4.5.79.3 Success conditions

ID	Condition
op	post: vdev.op == VDEV_OP_LOCK
comm_state	post: vdev.comm_state == DEV_COMM_PENDING

B4.5.79.4 Footprint

ID	Value
op	vdev.op

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Value
comm_state	`vdev.comm_state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.80 RMI_VDEV_START command

Start VDEV.

See also:

- A9.4.3 *Virtual device lifecycle*

B4.5.80.1 Interface

B4.5.80.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001D3`
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

B4.5.80.1.2 Context

The RMI_VDEV_START command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV

B4.5.80.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.80.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_state	`pre: GranuleAt(rd).state != GRAN_RD` `post: result.status == RMI_ERROR_INPUT`

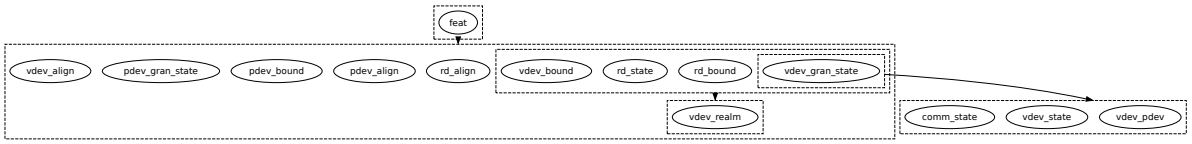
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsTracked(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev.realm != rd post: result.status == RMI_ERROR_INPUT
vdev_pdev	pre: vdev.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE
vdev_state	pre: vdev.vdev_state != VDEV_LOCKED post: result.status == RMI_ERROR_DEVICE
comm_state	pre: vdev.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_DEVICE

B4.5.80.2.1 Failure condition ordering

```
[rd_bound, rd_state, vdev_bound, vdev_gran_state] < [vdev_realm]
[feat] < [rd_align, rd_bound, rd_state, pdev_align, pdev_bound,
     pdev_gran_state, vdev_align, vdev_bound, vdev_gran_state,
     vdev_realm]
[vdev_gran_state] < [vdev_pdev, vdev_state, comm_state]
```



B4.5.80.3 Success conditions

ID	Condition
op	post: vdev.op == VDEV_OP_START
comm_state	post: vdev.comm_state == DEV_COMM_PENDING

B4.5.80.4 Footprint

ID	Value
op	vdev.op

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Value
comm_state	`vdev.comm_state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.81 RMI_VDEV_UNLOCK command

Unlock a VDEV.

See also:

- A9.4.3 *Virtual device lifecycle*

B4.5.81.1 Interface

B4.5.81.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400018A`
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

B4.5.81.1.2 Context

The RMI_VDEV_UNLOCK command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV
mapped	RmmVdevAddrResult	`VdevFindMapped(vdev)`	false	Result of scanning for mapped Granules

DRAFT

B4.5.81.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
addr	X1	63:0	Address	Address of assigned Granule

B4.5.81.2 Failure conditions

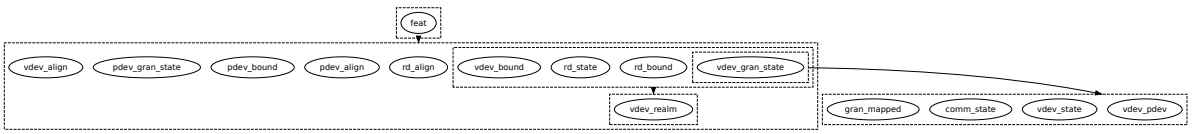
ID	Condition
feat	`pre: Rmm().static.feat_da != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsTracked(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev.realm != rd post: result.status == RMI_ERROR_INPUT
vdev_pdev	pre: vdev.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE
vdev_state	pre: (vdev.vdev_state != VDEV_LOCKED && vdev.vdev_state != VDEV_STARTED && vdev.vdev_state != VDEV_ERROR) post: result.status == RMI_ERROR_DEVICE
comm_state	pre: vdev.comm_state != DEV_COMM_IDLE post: result.status == RMI_ERROR_DEVICE
gran_mapped	pre: mapped.valid == RMM_TRUE post: (result.status == RMI_ERROR_GRANULE && addr == mapped.addr)

B4.5.81.2.1 Failure condition ordering

```
[rd_bound, rd_state, vdev_bound, vdev_gran_state] < [vdev_realm]
[feat] < [rd_align, rd_bound, rd_state, pdev_align, pdev_bound,
     pdev_gran_state, vdev_align, vdev_bound, vdev_gran_state,
     vdev_realm]
[vdev_gran_state] < [vdev_pdev, vdev_state, comm_state, gran_mapped]
```



B4.5.81.3 Success conditions

ID	Condition
dma_state	post: vdev.dma_state == VDEV_DMA_DISABLED
op	post: vdev.op == VDEV_OP_UNLOCK
comm_state	post: vdev.comm_state == DEV_COMM_PENDING

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.81.4 Footprint

ID	Value
op	`vdev.op`
comm_state	`vdev.comm_state`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.82 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*

DRAFT

B4.5.82.1 Interface

B4.5.82.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000150`
req	X1	63:0	RmiInterfaceVersion	Requested interface revision

B4.5.82.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
lower	X1	63:0	RmiInterfaceVersion	Lower supported interface revision
higher	X2	63:0	RmiInterfaceVersion	Higher supported interface revision

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.82.2 Failure conditions

ID	Condition
incompat_lower	`pre: (!RmiVersionIsSupported(req) && RmiVersionLowerIsSupported(req))` `post: (result.status == RMI_ERROR_INPUT && VersionEqual(lower, RmiVersionHighestBelow(req)) && VersionEqual(higher, RmiVersionHighest()))`
incompat_higher	`pre: (!RmiVersionIsSupported(req) && !RmiVersionLowerIsSupported(req) && RmiVersionHigherIsSupported(req))` `post: (result.status == RMI_ERROR_INPUT && VersionEqual(lower, higher) && VersionEqual(higher, RmiVersionHighest()))`

B4.5.82.2.1 Failure condition ordering

The RMI_VERSION command does not have any failure condition orderings.

B4.5.82.3 Success conditions

ID	Condition
lower	`post: VersionEqual(lower, req)`
higher	`post: VersionEqual(higher, RmiVersionHighest())`

B4.5.82.4 Footprint

The RMI_VERSION command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.83 RMI_VSMMU_CMD_COMPLETE command

Complete a command from the VSMMU.

See also:

- A9.8.6 *Page Request Interface events*
- B4.5.84 *RMI_VSMMU_CMD_GET command*

B4.5.83.1 Interface

B4.5.83.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400020D`
rd	X1	63:0	Address	PA of the RD
vsmmu_ptr	X2	63:0	Address	PA of the VSMMU
pdev_ptr	X3	63:0	Address	PA of the parent PDEV
vdev_ptr	X4	63:0	Address	PA of the VDEV

B4.5.83.1.2 Context

The RMI_VSMMU_CMD_COMPLETE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`RealmAt(rd)`	false	Realm
vsmmu	RmmVsmmu	`VsmmuAt(vsmmu_ptr)`	false	VSMMU
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV

B4.5.83.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
flags	X1	63:0	RmiVsmmuCmdFlags	Command flags
vsid	X2	63:0	Bits64	Virtual SMMU Stream ID
msi_addr	X3	63:0	Address	Address of virtual MSI to be injected into the Realm. This is valid if flags.irq == RMI_TRUE.
msi_data	X4	63:0	Bits64	Data of virtual MSI to be injected into the Realm. This is valid if flags.irq == RMI_TRUE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.83.2 Failure conditions

ID	Condition
feat	pre: Rmm().static.feat_vsmmu != FEATURE_TRUE post: result.status == RMI_ERROR_NOT_SUPPORTED
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm.state != REALM_NEW post: result.status == RMI_ERROR_REALM
vsmmu_align	pre: !AddrIsRmiGranuleAligned(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_bound	pre: !PaIsTracked(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_state	pre: GranuleAt(vsmmu_ptr).state != GRAN_VSMMU post: result.status == RMI_ERROR_INPUT
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsTracked(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev.realm != rd post: result.status == RMI_ERROR_INPUT
vdev_vsmmu	pre: vdev.vsmmu_addr != vsmmu_ptr post: result.status == RMI_ERROR_INPUT
vsmmu_realm	pre: vsmmu.realm != rd post: result.status == RMI_ERROR_INPUT
vdev_pdev	pre: vdev.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE

B4.5.83.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[feat] < [vsmmu_align, vsmmu_bound, vsmmu_state]
```

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.83.3 Success conditions

The RMI_VSMMU_CMD_COMPLETE command does not have any success conditions.

B4.5.83.4 Footprint

The RMI_VSMMU_CMD_COMPLETE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.84 RMI_VSMMU_CMD_GET command

Get a command from the VSMMU.

See also:

- A9.8.6 *Page Request Interface events*
- B4.5.83 *RMI_VSMMU_CMD_COMPLETE command*

B4.5.84.1 Interface

B4.5.84.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400020C`
vsmmu_ptr	X1	63:0	Address	PA of the VSMMU
rd	X2	63:0	Address	PA of the RD

B4.5.84.1.2 Context

The RMI_VSMMU_CMD_GET command operates on the following context.

Name	Type	Value	Before	Description
vsmmu	RmmVsmmu	`VsmmuAt(vsmmu_ptr)`	false	VSMMU
realm	RmmRealm	`RealmAt(rd)`	false	Realm

DRAFT

B4.5.84.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
flags	X1	63:0	RmiVsmmuCmdFlags	Command flags
vsid	X2	63:0	Bits64	Virtual SMMU Stream ID
msi_addr	X3	63:0	Address	Address of virtual MSI to be injected into the Realm. This is valid if flags.irq == RMI_TRUE.
msi_data	X4	63:0	Bits64	Data of virtual MSI to be injected into the Realm. This is valid if flags.irq == RMI_TRUE.

B4.5.84.2 Failure conditions

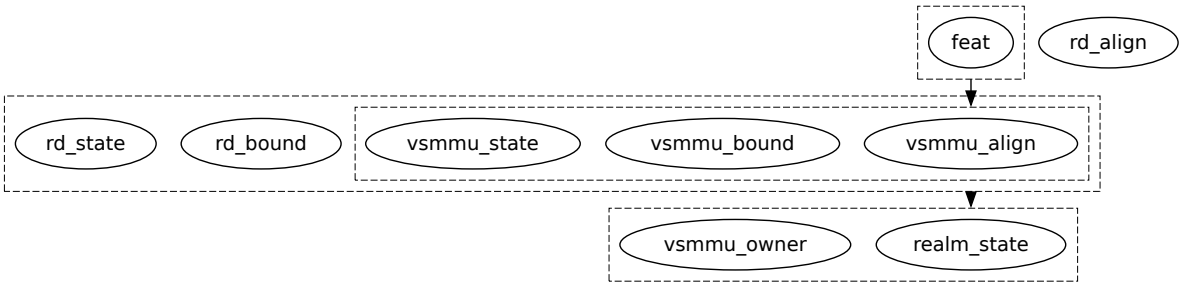
ID	Condition
feat	`pre: Rmm().static.feat_vsmmu != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
vsmmu_align	pre: !AddrIsRmiGranuleAligned(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_bound	pre: !PaIsTracked(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_state	pre: GranuleAt(vsmmu_ptr).state != GRAN_VSMMU post: result.status == RMI_ERROR_INPUT
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
vsmmu_owner	pre: vsmmu.realm != rd post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm.state != REALM_NEW post: result.status == RMI_ERROR_REALM

B4.5.84.2.1 Failure condition ordering

```
[vsmmu_align, vsmmu_bound, vsmmu_state, rd_bound, rd_state] <
     [realm_state, vsmmu_owner]
[feat] < [vsmmu_align, vsmmu_bound, vsmmu_state]
```



B4.5.84.3 Success conditions

The RMI_VSMMU_CMD_GET command does not have any success conditions.

B4.5.84.4 Footprint

The RMI_VSMMU_CMD_GET command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.85 RMI_VSMMU_CREATE command

Create a VSMMU.

The RMI_VSMMU_CREATE command may initiate a Stateful RMI Operation.

The RMI_VSMMU_CREATE command may initiate a memory-transferring RMI Operation.

See also:

- A9.8 *Virtual SMMU*
- B4.3.4 *Object creation and destruction*
- B4.5.86 *RMI_VSMMU_DESTROY command*

B4.5.85.1 Interface

B4.5.85.1.1 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

B4.5.85.1.2 Context

The RMI_VSMMU_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
vsmmu	RmmVsmmu	`VsmmuAt(vsmmu_ptr)`	false	VSMMU
params	RmiVsmmuParams	`RmiVsmmuParamsAt(params_ptr)`	false	VSMMU parameters

B4.5.85.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.85.2 Failure conditions

ID	Condition
feat	`pre: Rmm().static.feat_vsmmu != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`

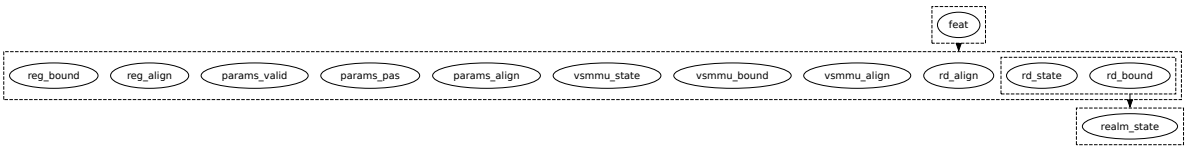
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm.state != REALM_NEW post: result.status == RMI_ERROR_REALM
vsmmu_align	pre: !AddrIsRmiGranuleAligned(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_bound	pre: !PaIsDelegableConventionalFine(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_state	pre: GranuleAt(vsmmu_ptr).state != GRAN_DELEGATED post: result.status == RMI_ERROR_INPUT
params_align	pre: !AddrIsRmiGranuleAligned(params_ptr) post: result.status == RMI_ERROR_INPUT
params_pas	pre: !NonSecureAccessPermitted(params_ptr) post: result.status == RMI_ERROR_INPUT
params_valid	pre: !RmiVsmmuParamsIsValid(params_ptr) post: result.status == RMI_ERROR_INPUT
reg_align	pre: (!AddrIsRmiGranuleAligned(params.reg_base) || !AddrIsRmiGranuleAligned(params.reg_top)) post: result.status == RMI_ERROR_INPUT
reg_bound	pre: (!AddrIsProtected(params.reg_base, realm) || !AddrIsProtected(params.reg_top, realm) || UInt(params.reg_top) <= UInt(params.reg_base)) post: result.status == RMI_ERROR_INPUT

B4.5.85.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[feat] < [rd_align, rd_bound, rd_state, vsmmu_align, vsmmu_bound,
     vsmmu_state, params_align, params_pas, params_valid, reg_align,
     reg_bound]
```



B4.5.85.3 Success conditions

ID	Condition
gran_state	post: GranuleAt(vsmmu_ptr).state == GRAN_VSMMU
state	post: vsmmu.state == VSMMU_INACTIVE
realm	post: vsmmu.realm == rd

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
reg_base	`post: vsmmu.reg_base == params.reg_base`
reg_top	`post: vsmmu.reg_top == params.reg_top`
aidr	`post: vsmmu.aidr == params.aidr`
idr	`post: (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]])`
num_vsmmus	`post: realm.num_vsmmus == realm_pre.num_vsmmus + 1`

B4.5.85.4 Footprint

ID	Value
state	`GranuleAt(vsmmu_ptr).state`
num_vsmmus	`realm.num_vsmmus`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.86 RMI_VSMMU_DESTROY command

Destroy a VSMMU.

The RMI_VSMMU_DESTROY command may initiate a Stateful RMI Operation.

The RMI_VSMMU_DESTROY command may initiate a memory-transferring RMI Operation.

See also:

- A9.8 *Virtual SMMU*
- B4.3.4 *Object creation and destruction*
- B4.5.85 *RMI_VSMMU_CREATE command*

B4.5.86.1 Interface

B4.5.86.1.1 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	PA of the VSMMU

B4.5.86.1.2 Context

The RMI_VSMMU_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
vsmmu	RmmVsmmu	`VsmmuAt(vsmmu_ptr)`	false	VSMMU

DRAFT

B4.5.86.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

B4.5.86.2 Failure conditions

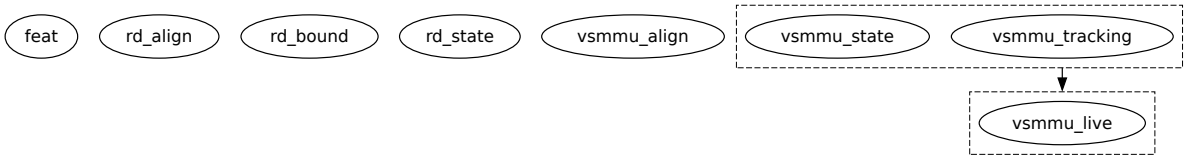
ID	Condition
feat	`pre: Rmm().static.feat_vsmmu != FEATURE_TRUE` `post: result.status == RMI_ERROR_NOT_SUPPORTED`
rd_align	`pre: !AddrIsRmiGranuleAligned(rd)` `post: result.status == RMI_ERROR_INPUT`
rd_bound	`pre: !PaIsTracked(rd)` `post: result.status == RMI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
vsmmu_align	pre: !AddrIsRmiGranuleAligned(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_tracking	pre: !PaIsTrackedFine(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_state	pre: GranuleAt(vsmmu_ptr).state != GRAN_VSMMU post: result.status == RMI_ERROR_INPUT
vsmmu_live	pre: VsmmuIsLive(vsmmu_ptr) post: result.status == RMI_ERROR_DEVICE

B4.5.86.2.1 Failure condition ordering

```
[vsmmu_tracking, vsmmu_state] < [vsmmu_live]
```



B4.5.86.3 Success conditions

ID	Condition
gran_state	post: GranuleAt(vsmmu_ptr).state == GRAN_DELEGATED
num_vsmmus	post: realm.num_vsmmus == realm_pre.num_vsmmus - 1

B4.5.86.4 Footprint

ID	Value
state	GranuleAt(vsmmu_ptr).state
num_vsmmus	realm.num_vsmmus

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.87 RMI_VSMMU_EVENT_HANDLE command

Notify VSMMU of a pending event.

See also:

- A9.8.6 *Page Request Interface events*

B4.5.87.1 Interface

B4.5.87.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001D6`
psmmu_ptr	X1	63:0	Address	PA of the PSMMU
rd	X2	63:0	Address	PA of the RD
vsmmu_ptr	X3	63:0	Address	PA of the VSMMU
pdev_ptr	X4	63:0	Address	PA of the PDEV
vdev_ptr	X5	63:0	Address	PA of the VDEV

B4.5.87.1.2 Context

The RMI_VSMMU_EVENT_HANDLE command operates on the following context.

Name	Type	Value	Before	Description
psmmu	RmmPsmmu	`PsmmuAt(psmmu_ptr)`	false	PSMMU
realm	RmmRealm	`RealmAt(rd)`	false	Realm
vsmmu	RmmVsmmu	`VsmmuAt(vsmmu_ptr)`	false	VSMMU
pdev	RmmPdev	`PdevAt(pdev_ptr)`	false	PDEV
vdev	RmmVdev	`VdevAt(vdev_ptr)`	false	VDEV

B4.5.87.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result
flags	X1	63:0	RmiVsmmuEventFlags	Action required by Host
msi_addr	X2	63:0	Address	Address of virtual MSI to be injected into the Realm. This is valid if flags.irq == RMI_TRUE.
msi_data	X3	63:0	Bits64	Data of virtual MSI to be injected into the Realm. This is valid if flags.irq == RMI_TRUE.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Register	Bits	Type	Description
fipa	X4	63:0	Address	Faulting IPA. This is valid if flags.abort == RMI_TRUE.
syndrome	X5	63:0	Bits64	RnW[0], S2[1] and Class[3:2] attributes, as specified by SMMU. This is valid if flags.abort == RMI_TRUE.

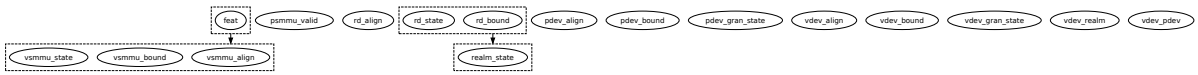
B4.5.87.2 Failure conditions

ID	Condition
feat	pre: Rmm().static.feat_vsmmu != FEATURE_TRUE post: result.status == RMI_ERROR_NOT_SUPPORTED
psmmu_valid	pre: !PsmmuAddrIsValid(psmmu_ptr) post: result.status == RMI_ERROR_INPUT
rd_align	pre: !AddrIsRmiGranuleAligned(rd) post: result.status == RMI_ERROR_INPUT
rd_bound	pre: !PaIsTracked(rd) post: result.status == RMI_ERROR_INPUT
rd_state	pre: GranuleAt(rd).state != GRAN_RD post: result.status == RMI_ERROR_INPUT
realm_state	pre: realm.state != REALM_NEW post: result.status == RMI_ERROR_REALM
vsmmu_align	pre: !AddrIsRmiGranuleAligned(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_bound	pre: !PaIsTracked(vsmmu_ptr) post: result.status == RMI_ERROR_INPUT
vsmmu_state	pre: GranuleAt(vsmmu_ptr).state != GRAN_VSMMU post: result.status == RMI_ERROR_INPUT
pdev_align	pre: !AddrIsRmiGranuleAligned(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_bound	pre: !PaIsTracked(pdev_ptr) post: result.status == RMI_ERROR_INPUT
pdev_gran_state	pre: GranuleAt(pdev_ptr).state != GRAN_PDEV post: result.status == RMI_ERROR_INPUT
vdev_align	pre: !AddrIsRmiGranuleAligned(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_bound	pre: !PaIsTracked(vdev_ptr) post: result.status == RMI_ERROR_INPUT
vdev_gran_state	pre: GranuleAt(vdev_ptr).state != GRAN_VDEV post: result.status == RMI_ERROR_INPUT
vdev_realm	pre: vdev.realm != rd post: result.status == RMI_ERROR_INPUT
vdev_pdev	pre: vdev.pdev != pdev_ptr post: result.status == RMI_ERROR_DEVICE

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.87.2.1 Failure condition ordering

```
[rd_bound, rd_state] < [realm_state]
[feat] < [vsmmu_align, vsmmu_bound, vsmmu_state]
```



B4.5.87.3 Success conditions

ID	Condition

gerr_irq	`pre: VSMMU has asserted GERROR interrupt.`
	`post: (flags.irq == RMI_TRUE`
	`&& msi_addr == vsmmu.msi_config.gerr_addr`
	`&& msi_data == vsmmu.msi_config.gerr_data)`

priq_irq	`pre: VSMMU has asserted PRIQ interrupt.`
	`post: (flags.irq == RMI_TRUE`
	`&& msi_addr == vsmmu.msi_config.priq_addr`
	`&& msi_data == vsmmu.msi_config.priq_data)`

B4.5.87.4 Footprint

The RMI_VSMMU_EVENT_HANDLE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.5.88 RMI_VSMMU_FEATURES command

Returns VSMMU features supporte by the RMM.

B4.5.88.1 Interface

B4.5.88.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400020B`
features_ptr	X1	63:0	Address	PA of the VSMMU features structure

B4.5.88.1.2 Context

The RMI_VSMMU_FEATURES command operates on the following context.

Name	Type	Value	Before	Description
features	RmiVsmmuFeatures	`VsmmuFeaturesAt(` `↪features_ptr)`	false	VSMMU features

B4.5.88.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RmiResult	Command result

DRAFT

B4.5.88.2 Failure conditions

ID	Condition
features_align	`pre: !AddrIsAligned(features_ptr, 0x100)` `post: result.status == RMI_ERROR_INPUT`
features_pas	`pre: !NonSecureAccessPermitted(features_ptr)` `post: result.status == RMI_ERROR_INPUT`

B4.5.88.2.1 Failure condition ordering

The RMI_VSMMU_FEATURES command does not have any failure condition orderings.

B4.5.88.3 Success conditions

ID	Condition
features	`post: features is populated with VSMMU features supported by the RMM.`

B4.5.88.4 Footprint

The RMI_VSMMU_FEATURES command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6 RMI types

This section defines types which are used in the RMI interface.

B4.6.1 RmiAddrBlockSize type

The RmiAddrBlockSize enumeration represents size of a block of contiguous address space.

The RmiAddrBlockSize enumeration is a concrete type.

The width of the RmiAddrBlockSize enumeration is 2 bits.

The values of the RmiAddrBlockSize enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_PAGE_L3	Level 3 page size.
1	RMI_BLOCK_L2	Level 2 block size.
2	RMI_BLOCK_L1	Level 1 block size.
3	RMI_BLOCK_L0	Level 0 block size.

The RmiAddrBlockSize enumeration is used in the following types:

- RmiAddrSetDesc
- RmiOpMemDonateReq
- RmiAddrRangeDesc64KB
- RmiAddrRangeDesc16KB
- RmiAddrRangeDesc
- RmiAddrRangeDesc4KB

DRAFT

B4.6.2 RmiAddrRange type

The RmiAddrRange structure contains address range.

The RmiAddrRange structure is a concrete type.

The width of the RmiAddrRange structure is 16 (`0x10`) bytes.

The members of the RmiAddrRange 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 RmiAddrRange structure is used in the following types:

- RmiVdevParams
- RmiPdevStreamParams

B4.6.3 RmiAddrRangeDesc type

The RmiAddrRangeDesc fieldset contains a descriptor which identifies a contiguous address range.

The RmiAddrRangeDesc fieldset is a concrete type.

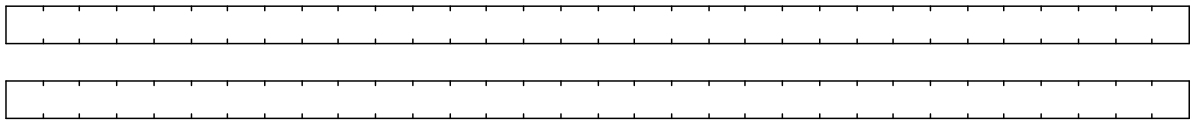
Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The width of the RmiAddrRangeDesc fieldset is 64 bits.

See also:

- B4.4.1 *RMI Address Range Descriptor*

The fields of the RmiAddrRangeDesc fieldset are shown in the following diagram.



The fields of the RmiAddrRangeDesc fieldset are shown in the following table.

Name	Bits	Description	Value
data	63:0	Conditional	See below

Encodings for field data:

Name	Condition	Fieldset
granule_4kb	`RmmGlobal().static.rmm_granule_size == RMI_GRANULE_SIZE_4KB`	RmiAddrRangeDesc4KB
granule_16kb	`RmmGlobal().static.rmm_granule_size == RMI_GRANULE_SIZE_16KB`	RmiAddrRangeDesc16KB
granule_64kb	`RmmGlobal().static.rmm_granule_size == RMI_GRANULE_SIZE_64KB`	RmiAddrRangeDesc64KB

The RmiAddrRangeDesc fieldset is used in the following types:

- RmiAddrSetDesc

B4.6.4 RmiAddrRangeDesc16KB type

The RmiAddrRangeDesc16KB fieldset contains a descriptor which identifies a contiguous address range, when RMI Granule Size is 16KB.

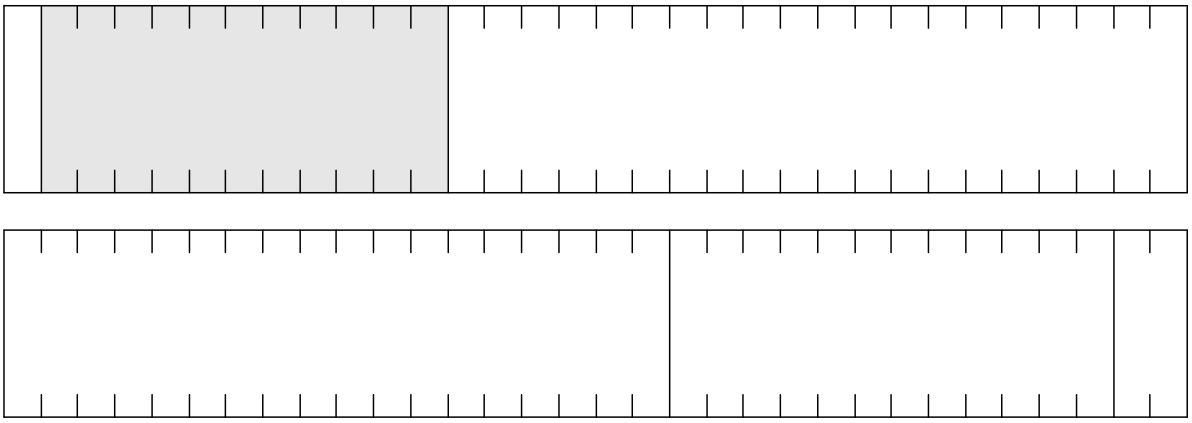
The RmiAddrRangeDesc16KB fieldset is a concrete type.

The width of the RmiAddrRangeDesc16KB fieldset is 64 bits.

The fields of the RmiAddrRangeDesc16KB fieldset are shown in the following diagram.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiAddrRangeDesc16KB fieldset are shown in the following table.

Name	Bits	Description	Value
size	1:0	Size of each block in the range	RmiAddrBlockSize
count	13:2	Number of blocks in the range	UInt12
addr	51:14	Base address of the range	Bits38
	62:52	Reserved	SBZ
state	63	State of memory within the range	RmiOpMemState

The RmiAddrRangeDesc16KB fieldset is used in the following types:

- RmiAddrSetDesc
- RmiAddrRangeDesc

DRAFT

B4.6.5 RmiAddrRangeDesc4KB type

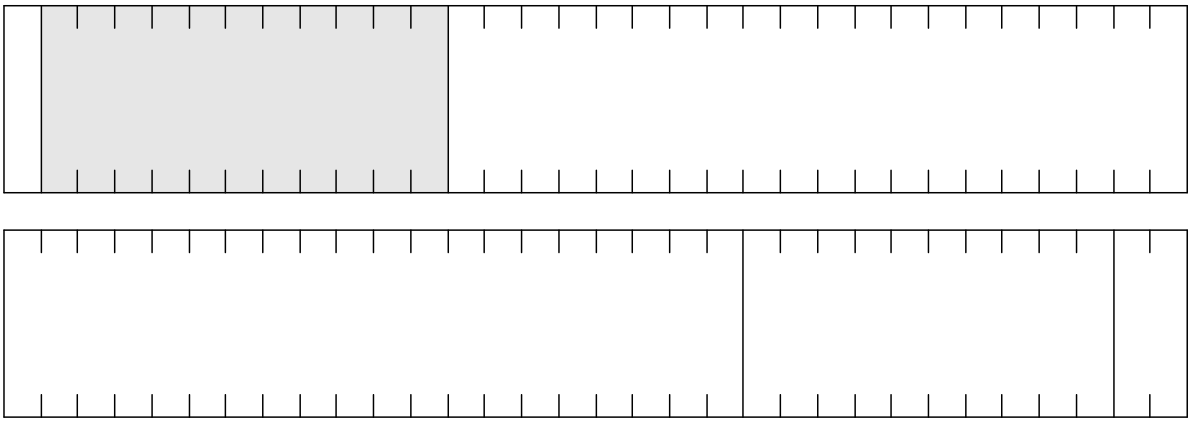
The RmiAddrRangeDesc4KB fieldset contains a descriptor which identifies a contiguous address range, when RMI Granule Size is 4KB.

The RmiAddrRangeDesc4KB fieldset is a concrete type.

The width of the RmiAddrRangeDesc4KB fieldset is 64 bits.

The fields of the RmiAddrRangeDesc4KB fieldset are shown in the following diagram.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiAddrRangeDesc4KB fieldset are shown in the following table.

Name	Bits	Description	Value
size	1:0	Size of each block in the range	RmiAddrBlockSize
count	11:2	Number of blocks in the range	UInt10
addr	51:12	Base address of the range	Bits40
	62:52	Reserved	SBZ
state	63	State of memory within the range	RmiOpMemState

The RmiAddrRangeDesc4KB fieldset is used in the following types:

- RmiAddrSetDesc
- RmiAddrRangeDesc

DRAFT

B4.6.6 RmiAddrRangeDesc64KB type

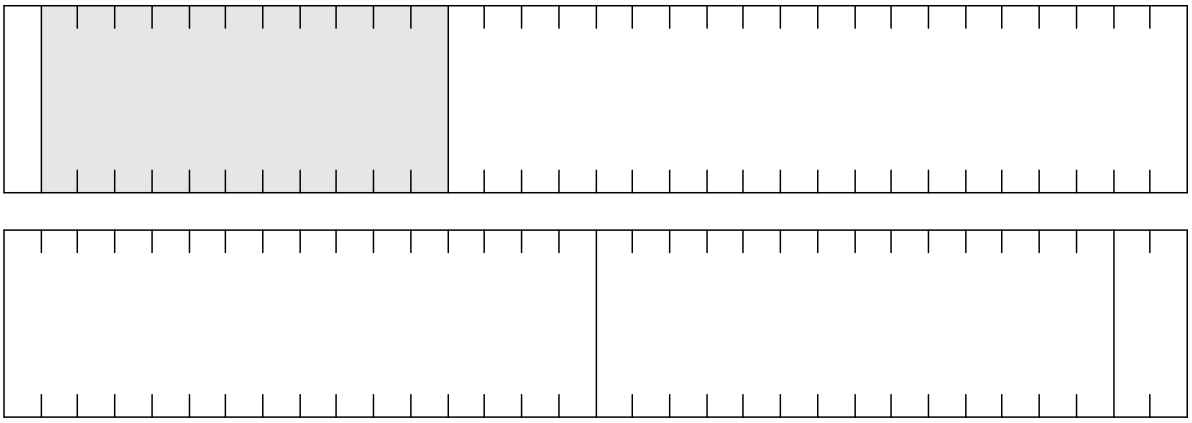
The RmiAddrRangeDesc64KB fieldset contains a descriptor which identifies a contiguous address range, when RMI Granule Size is 64KB.

The RmiAddrRangeDesc64KB fieldset is a concrete type.

The width of the RmiAddrRangeDesc64KB fieldset is 64 bits.

The fields of the RmiAddrRangeDesc64KB fieldset are shown in the following diagram.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiAddrRangeDesc64KB fieldset are shown in the following table.

Name	Bits	Description	Value
size	1:0	Size of each block in the range	RmiAddrBlockSize
count	15:2	Number of blocks in the range	UInt14
addr	51:16	Base address of the range	Bits36
	62:52	Reserved	SBZ
state	63	State of memory within the range	RmiOpMemState

The RmiAddrRangeDesc64KB fieldset is used in the following types:

- RmiAddrSetDesc
- RmiAddrRangeDesc

DRAFT

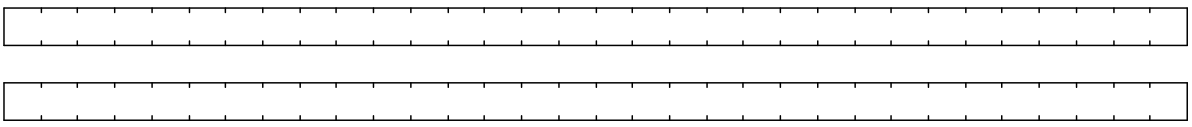
B4.6.7 RmiAddrSetDesc type

The RmiAddrSetDesc fieldset contains a descriptor which identifies an address set.

The RmiAddrSetDesc fieldset is a concrete type.

The width of the RmiAddrSetDesc fieldset is 64 bits.

The fields of the RmiAddrSetDesc fieldset are shown in the following diagram.



The fields of the RmiAddrSetDesc fieldset are shown in the following table.

Name	Bits	Description	Value
data	63:0	Conditional	See below

Encodings for field data:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Condition	Fieldset
single	`flags.oaddr_type == RMI_ADDR_TYPE_SINGLE`	RmiAddrRangeDesc
list_addr	`flags.oaddr_type == RMI_ADDR_TYPE_LIST`	RmiAddrSetList

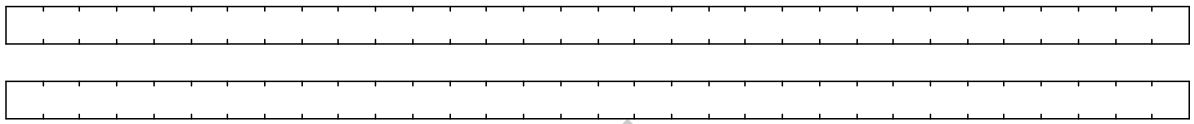
B4.6.8 RmiAddrSetList type

The RmiAddrSetList fieldset contains address of a Granule which contains a list of RMI Address Range Descriptors.

The RmiAddrSetList fieldset is a concrete type.

The width of the RmiAddrSetList fieldset is 64 bits.

The fields of the RmiAddrSetList fieldset are shown in the following diagram.



The fields of the RmiAddrSetList fieldset are shown in the following table.

Name	Bits	Description	Value
addr	63:0	Address of the Granule	Address

The RmiAddrSetList fieldset is used in the following types:

- RmiAddrSetDesc

B4.6.9 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:

- RmiVsmmuEventFlags
- RmiPsmmuIrqSet
- RmiDevCommExitFlags
- RmiVsmmuCmdFlags
- RmiPsmmuIrqResult

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.10 RmiContinueBeyond type

The RmiContinueBeyond enumeration represents how RMI_OP_CONTINUE should behave once it has transitioned all objects in the target set from the intermediate state to the destination state.

The RmiContinueBeyond enumeration is a concrete type.

The width of the RmiContinueBeyond enumeration is 1 bits.

The values of the RmiContinueBeyond enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_CONTINUE_KEEP_GOING	Continue processing the target set.
1	RMI_CONTINUE_STOP	Return to the Host.

The RmiContinueBeyond enumeration is used in the following types:

- RmiContinueFlags

B4.6.11 RmiContinueFlags type

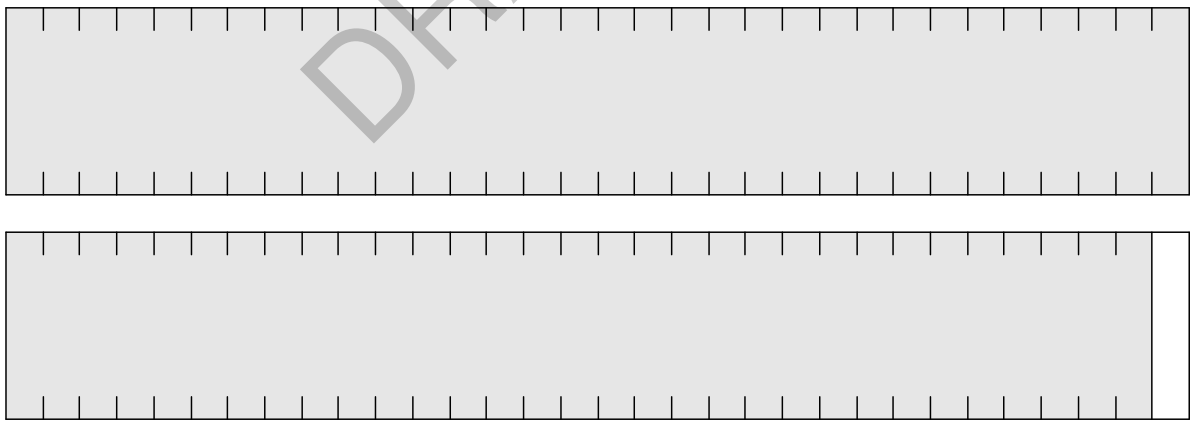
The RmiContinueFlags fieldset contains flags provided by the Host when continuing an incomplete operation.

The RmiContinueFlags fieldset is a concrete type.

The width of the RmiContinueFlags fieldset is 64 bits.

The fields of the RmiContinueFlags fieldset are shown in the following diagram.

DRAFT



The fields of the RmiContinueFlags fieldset are shown in the following table.

Name	Bits	Description	Value
beyond	0	How RMI_OP_CONTINUE should behave once it has transitioned all objects in the target set from the intermediate state to the destination state	RmiContinueBeyond
	63:1	Reserved	SBZ

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

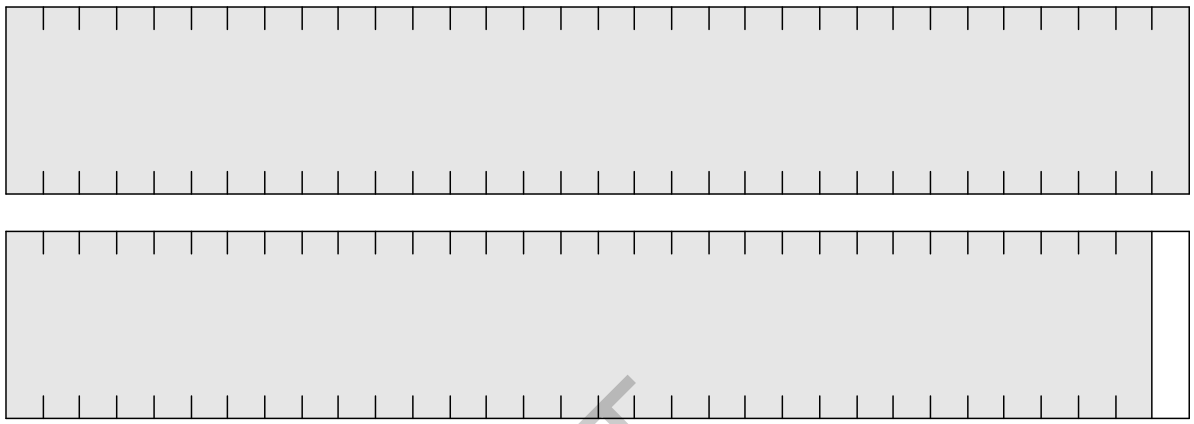
B4.6.12 RmiDataFlags type

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

DRAFT

B4.6.13 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

B4.6.14 RmiDevCommData type

The RmiDevCommData structure contains data structure shared between Host and RMM for device communication.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

B4.6.15 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:

- A9.5.2.2 *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
rsp_addr	`0x10`	Address	Address of response buffer
rsp_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

B4.6.16 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:

- A9.5.2.1 *Device communication exit data structure*

The members of the RmiDevCommExit structure are shown in the following table.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
flags	0x0	RmiDevCommExitFlags	Flags indicating action(s) which the Host is requested to perform
req_cache_offset	0x8	UInt64	If flags.req_cache is true, offset in the device request buffer to the start of data to be cached, in bytes
req_cache_len	0x10	UInt64	If flags.req_cache is true, amount of device request data to be cached, in bytes
rsp_cache_offset	0x18	UInt64	If flags.rsp_cache is true, offset in the device response buffer to the start of data to be cached, in bytes
rsp_cache_len	0x20	UInt64	If flags.rsp_cache is true, amount of device response data to be cached, in bytes
cache_object_id	0x28	RmiDevCommObject	If flags.req_cache is true and / or flags.rsp_cache is true, identifier for the object to be cached
protocol	0x30	RmiDevCommProtocol	If flags.req_send is true, protocol to use
req_delay	0x38	UInt64	If flags.req_send is true, amount of time to wait before sending the request, in microseconds.
req_len	0x40	UInt64	If flags.req_send is true, amount of valid data in request buffer in bytes
rsp_timeout	0x48	UInt64	Amount of time to wait (measured from the most recent exit which had flags.rsp_reset = true) for device response in microseconds.

DRAFT

Unused bits of the RmiDevCommExit structure MBZ.

The RmiDevCommExit structure is used in the following types:

- RmiDevCommData

B4.6.17 RmiDevCommExitFlags type

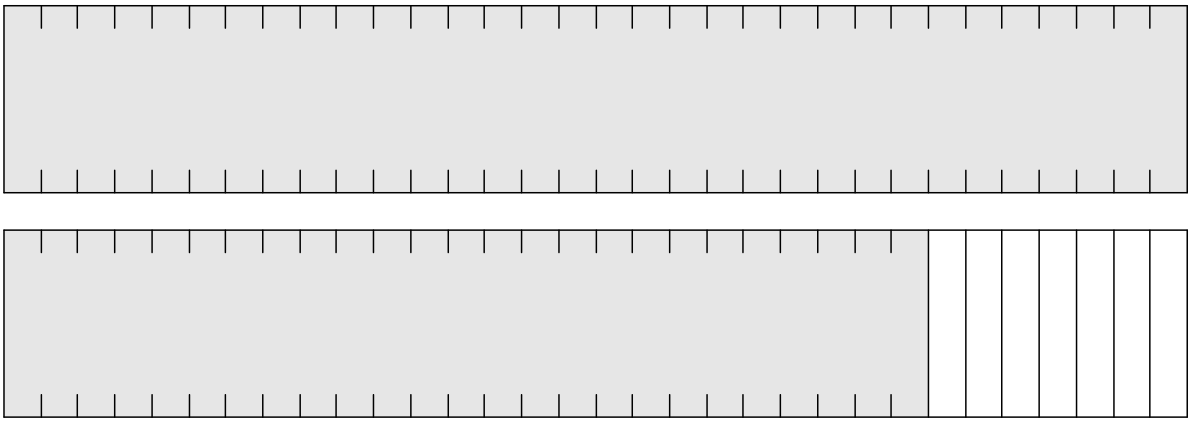
The RmiDevCommExitFlags fieldset contains flags provided by the RMM during a device transaction.

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiDevCommExitFlags fieldset are shown in the following table.

Name	Bits	Description	Value
req_cache	0	Whether the Host is requested to cache data from the device request buffer	RmiBoolean
rsp_cache	1	Whether the Host is requested to cache data from the device response buffer	RmiBoolean
req_send	2	Whether the Host is requested to send data from the device request buffer to the device	RmiBoolean
rsp_wait	3	Whether the RMM is waiting for a response from the device	RmiBoolean
rsp_reset	4	Whether to reset the response timer	RmiBoolean
multi	5	Whether the device transaction contains more than one (device request, device response) tuple	RmiBoolean
stream_wait	6	Whether a transaction associated with another device, to which this device is connected via a PDEV stream, must proceed before this transaction can proceed	RmiBoolean
	63:7	Reserved	MBZ

DRAFT

The RmiDevCommExitFlags fieldset is used in the following types:

- RmiDevCommExit

B4.6.18 RmiDevCommObject type

The RmiDevCommObject enumeration represents identifier of a device communication object which the Host is requested to cache.

The RmiDevCommObject enumeration is a concrete type.

The width of the RmiDevCommObject enumeration is 8 bits.

The values of the RmiDevCommObject enumeration are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
0	RMI_DEV_PROTOCOL_DATA	Protocol data associated with a PDEV
1	RMI_DEV_IDENTITY	Device identity associated with a PDEV
2	RMI_DEV_MEASUREMENTS	Device measurements associated with a VDEV
3	RMI_DEV_INTERFACE_REPORT	Device interface report associated with a VDEV

Unused encodings for the RmiDevCommObject enumeration are reserved for use by future versions of this specification.

The RmiDevCommObject enumeration is used in the following types:

- RmiDevCommExit

B4.6.19 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)* [21]
1	RMI_PROTOCOL_SECURE_SPDM	Secure SPDM See *Secured Messages using SPDM Specification version 1.1.0* [17]

DRAFT

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

B4.6.20 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
0	RMI_DEV_COMM_NONE	No device response has been received from the device. Either: • 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: • 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

B4.6.21 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.

DRAFT

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

B4.6.22 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	• During discovery: Feature is not supported. • During selection: Feature is not enabled.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
1	RMI_FEATURE_TRUE	• During discovery: Feature is supported. • During selection: Feature is enabled.

The RmiFeature enumeration is used in the following types:

- RmiPsmmuInfoFlags
- RmiPsmmuFlags
- RmiVdevFlags
- RmiFeatureRegister0
- RmiFeatureRegister3
- RmiRealmFlags0
- RmiFeatureRegister2
- RmiRealmFlags1
- RmiFeatureRegister1

B4.6.23 RmiFeatureRegister0 type

The RmiFeatureRegister0 fieldset contains RMI feature register 0.

The RmiFeatureRegister0 fieldset is a concrete type.

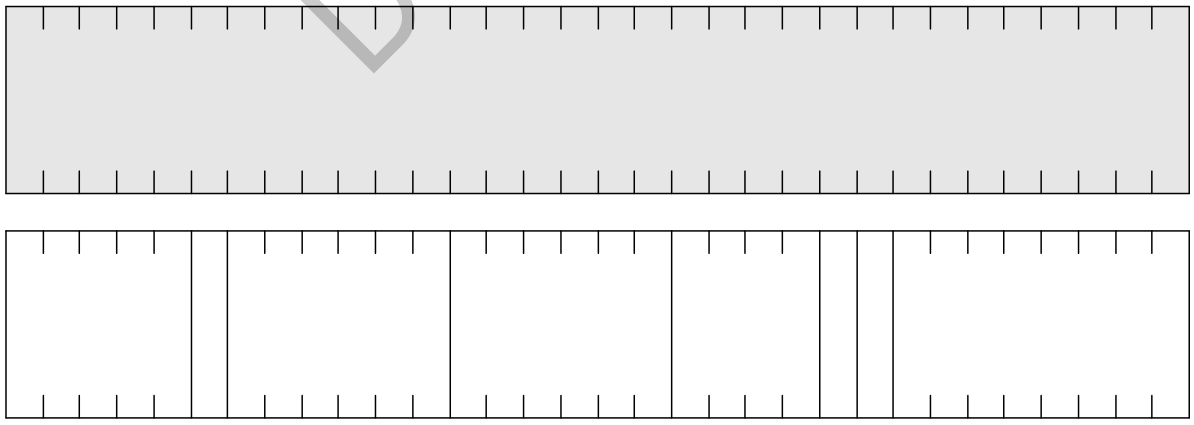
The width of the RmiFeatureRegister0 fieldset is 64 bits.

See also:

- Chapter A3 *Feature discovery and configuration*
- B4.5.5 *RMI_FEATURES command*

The fields of the RmiFeatureRegister0 fieldset are shown in the following diagram.

DRAFT



The fields of the RmiFeatureRegister0 fieldset are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 `VTCR_EL2.T0SZ` 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 `(SVE_VL + 1)*128`, similar to the value of `ZCR_ELx.LEN`.	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
	63:32	Reserved	MBZ

DRAFT

B4.6.24 RmiFeatureRegister1 type

The RmiFeatureRegister1 fieldset contains RMI feature register 1.

The RmiFeatureRegister1 fieldset is a concrete type.

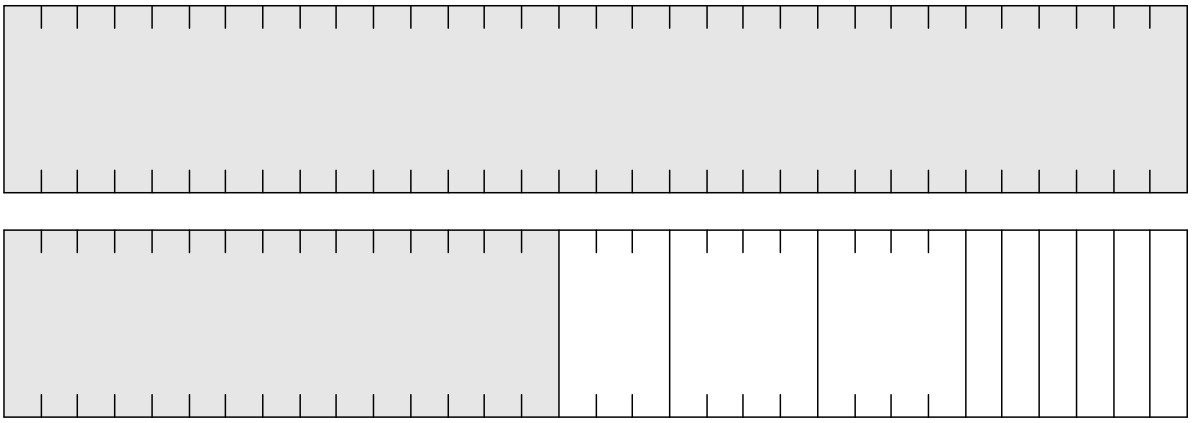
The width of the RmiFeatureRegister1 fieldset is 64 bits.

See also:

- Chapter A3 *Feature discovery and configuration*
- Chapter A11 *Realm memory encryption*
- B4.5.5 *RMI_FEATURES command*

The fields of the RmiFeatureRegister1 fieldset are shown in the following diagram.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiFeatureRegister1 fieldset are shown in the following table.

Name	Bits	Description	Value
RMI_GRAN_SZ_4KB	0	Whether an RMI Granule size of 4KB is supported	RmiFeature
RMI_GRAN_SZ_16KB	1	Whether an RMI Granule size of 16KB is supported	RmiFeature
RMI_GRAN_SZ_64KB	2	Whether an RMI Granule size of 64KB is supported	RmiFeature
HASH_SHA_256	3	Whether SHA-256 is supported	RmiFeature
HASH_SHA_384	4	Whether SHA-384 is supported	RmiFeature
HASH_SHA_512	5	Whether SHA-512 is supported	RmiFeature
MAX_RECS_ORDER	9:6	Order of the maximum number of RECs which can be created per Realm. The maximum number of RECs is computed as follows: `MAX_RECS = (2 ^ MAX_RECS_ORDER)- 1`	UInt4
L0GPTSZ	13:10	Value of Level 0 GPT entry size as encoded in GPCCR_EL3.L0GPTSZ.	Bits4
PPS	16:14	Value of Protected Physical Address Size as encoded in GPCCR_EL3.PPS.	Bits3
	63:17	Reserved	MBZ

DRAFT

B4.6.25 RmiFeatureRegister2 type

The RmiFeatureRegister2 fieldset contains RMI feature register 2.

The RmiFeatureRegister2 fieldset is a concrete type.

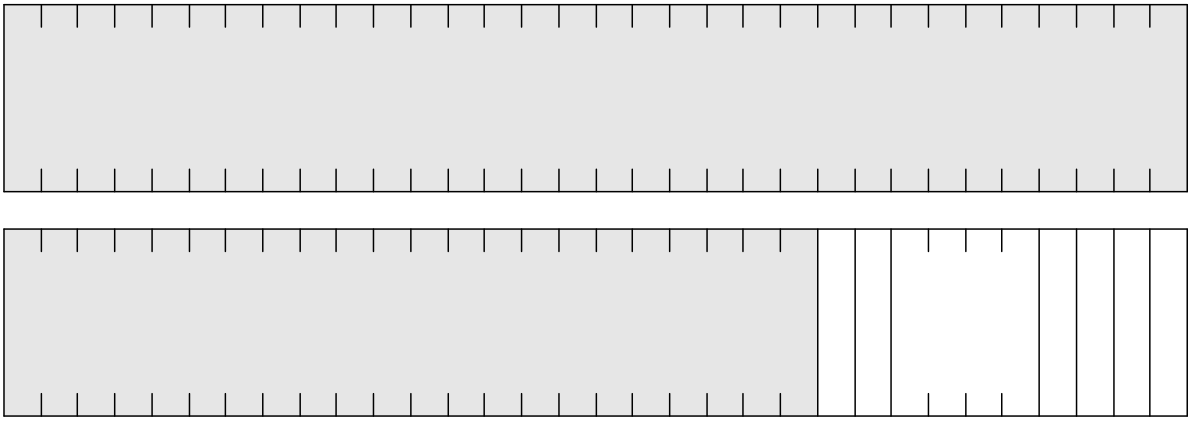
The width of the RmiFeatureRegister2 fieldset is 64 bits.

See also:

- Chapter A3 *Feature discovery and configuration*
- B4.5.5 *RMI_FEATURES command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The fields of the RmiFeatureRegister2 fieldset are shown in the following diagram.



The fields of the RmiFeatureRegister2 fieldset are shown in the following table.

Name	Bits	Description	Value
DA	0	Whether Realm device assignment is supported	RmiFeature
DA_COH	1	Whether coherent device assignment is supported. If DA == RMI_FEATURE_FALSE, this flag should also be RMI_FEATURE_FALSE.	RmiFeature
VSMMU	2	Whether virtual SMMU is supported. If DA == RMI_FEATURE_FALSE, this flag should also be RMI_FEATURE_FALSE.	RmiFeature
ATS	3	Whether ATS is supported. If DA == RMI_FEATURE_FALSE, this flag should also be RMI_FEATURE_FALSE.	RmiFeature
MAX_VDEVS_ORDER	7:4	Order of the maximum number of VDEVs which can be created per PDEV. The maximum number of VDEVs is computed as follows: `MAX_VDEVS = (2^MAX_VDEVS_ORDER)- 1`	UInt4
VDEV_KROU	8	Whether VDEV key refresh on unlock is required	RmiFeature
NON_TEE_STREAM	9	Whether NON_TEE PDEV stream type is supported	RmiFeature
	63:10	Reserved	MBZ

DRAFT

B4.6.26 RmiFeatureRegister3 type

The RmiFeatureRegister3 fieldset contains RMI feature register 3.

The RmiFeatureRegister3 fieldset is a concrete type.

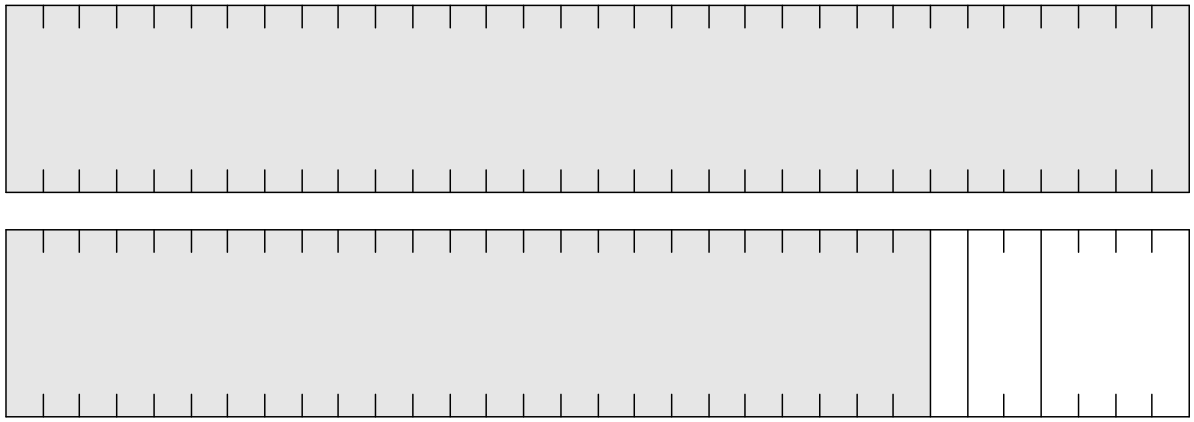
The width of the RmiFeatureRegister3 fieldset is 64 bits.

See also:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- Chapter A3 *Feature discovery and configuration*
- B4.5.5 *RMI_FEATURES command*

The fields of the RmiFeatureRegister3 fieldset are shown in the following diagram.



The fields of the RmiFeatureRegister3 fieldset are shown in the following table.

Name	Bits	Description	Value
MAX_NUM_AUX_PLANES	3:0	Maximum number of auxiliary Planes	UInt4
RTT_PLANE	5:4	RTT usage models supported for multi-Plane Realms. If only a single Plane is supported (that is, MAX_NUM_AUX_PLANES is 0), this field is RES0.	RmiRttPlaneFeature
RTT_S2AP_INDIRECT	6	Whether S2AP indirect encoding is supported	RmiFeature
	63:7	Reserved	MBZ

DRAFT

B4.6.27 RmiFeatureRegister4 type

The RmiFeatureRegister4 fieldset contains RMI feature register 4.

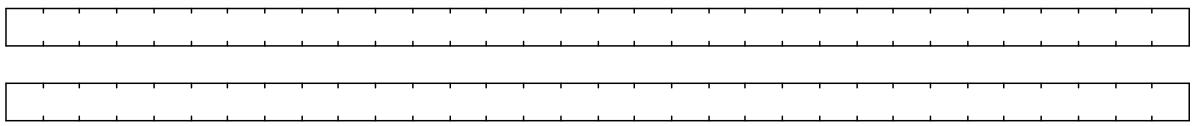
The RmiFeatureRegister4 fieldset is a concrete type.

The width of the RmiFeatureRegister4 fieldset is 64 bits.

See also:

- Chapter A3 *Feature discovery and configuration*
- B4.5.5 *RMI_FEATURES command*

The fields of the RmiFeatureRegister4 fieldset are shown in the following diagram.



The fields of the RmiFeatureRegister4 fieldset are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Bits	Description	Value
MEC_COUNT	63:0	Number of MECs.	UInt64

B4.6.28 RmiForceP0 type

The RmiForceP0 enumeration represents whether to force control to return Plane 0.

The RmiForceP0 enumeration is a concrete type.

The width of the RmiForceP0 enumeration is 1 bits.

The values of the RmiForceP0 enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_NO_FORCE_P0	Do not affect the Plane to which control is returned.
1	RMI_FORCE_P0	Force control to return to Plane 0.

The RmiForceP0 enumeration is used in the following types:

- RmiRecEnterFlags

B4.6.29 RmiGranuleSize type

The RmiGranuleSize enumeration represents a Granule size.

The RmiGranuleSize enumeration is a concrete type.

The width of the RmiGranuleSize enumeration is 8 bits.

The values of the RmiGranuleSize enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_GRANULE_SIZE_4KB	4KB.
1	RMI_GRANULE_SIZE_16KB	16KB.
2	RMI_GRANULE_SIZE_64KB	64KB.

Unused encodings for the RmiGranuleSize enumeration are reserved for use by future versions of this specification.

The RmiGranuleSize enumeration is used in the following types:

- RmiRmmConfig

B4.6.30 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
0	RMI_HASH_SHA_256	SHA-256 (*Secure Hash Standard (SHS)* [25])
1	RMI_HASH_SHA_512	SHA-512 (*Secure Hash Standard (SHS)* [25])
2	RMI_HASH_SHA_384	SHA-384 (*Secure Hash Standard (SHS)* [25])

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

B4.6.31 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.
1	RMI_INJECT_SEA	Inject an SEA into the Realm.

DRAFT

The RmiInjectSea enumeration is used in the following types:

- RmiRecEnterFlags

B4.6.32 RmiInterfaceVersion type

The RmiInterfaceVersion fieldset contains an RMI interface version.

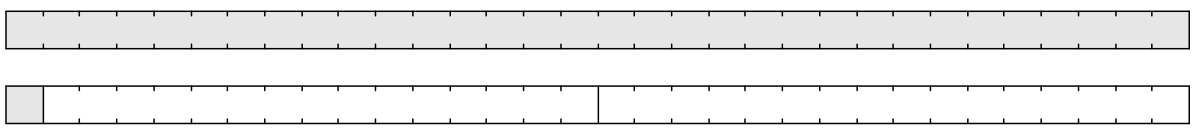
The RmiInterfaceVersion fieldset is a concrete type.

The width of the RmiInterfaceVersion fieldset is 64 bits.

See also:

- B4.1 *RMI version*
- B4.5.82 *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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

B4.6.33 RmiIrqCfg type

The RmiIrqCfg enumeration represents whether an IRQ is disabled, wired or MSI.

The RmiIrqCfg enumeration is a concrete type.

The width of the RmiIrqCfg enumeration is 2 bits.

The values of the RmiIrqCfg enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_IRQ_DISABLED	• Interrupt is disabled
1	RMI_IRQ_WIRED	• Interrupt is of Wired type
2	RMI_IRQ_MSI	• Interrupt is of MSI type

Unused encodings for the RmiIrqCfg enumeration are reserved for use by future versions of this specification.

The RmiIrqCfg enumeration is used in the following types:

- RmiPsmmuInfoFlags
- RmiPsmmuFlags

DRAFT

B4.6.34 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.
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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.35 RmiMecPolicy type

The RmiMecPolicy enumeration represents a MEC policy.

The RmiMecPolicy enumeration is a concrete type.

The width of the RmiMecPolicy enumeration is 2 bits.

The values of the RmiMecPolicy enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_MEC_POLICY_SHARED	The MEC protects memory owned by multiple Realms. A MEC with this policy may be referred to as a *Shared MEC*.
1	RMI_MEC_POLICY_PRIVATE	The MEC protects memory owned by a single Realm. A MEC with this policy may be referred to as a *Private MEC*.

Unused encodings for the RmiMecPolicy enumeration are reserved for use by future versions of this specification.

The RmiMecPolicy enumeration is used in the following types:

- RmiRealmFlags0

B4.6.36 RmiMemCategory type

The RmiMemCategory enumeration represents memory category.

The RmiMemCategory enumeration is a concrete type.

The width of the RmiMemCategory enumeration is 2 bits.

The values of the RmiMemCategory enumeration are shown in the following table.

DRAFT

Encoding	Name	Description
0	RMI_MEM_CATEGORY_CONVENTIONAL	Conventional memory.
1	RMI_MEM_CATEGORY_DEV_NCOH	Device non-coherent memory.
2	RMI_MEM_CATEGORY_DEV_COH	Device coherent memory.

Unused encodings for the RmiMemCategory enumeration are reserved for use by future versions of this specification.

B4.6.37 RmiOpCanCancel type

The RmiOpCanCancel enumeration represents whether RMI operation can be cancelled.

The RmiOpCanCancel enumeration is a concrete type.

The width of the RmiOpCanCancel enumeration is 1 bits.

The values of the RmiOpCanCancel enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_OP_CANNOT_CANCEL	Operation cannot be cancelled.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
1	RMI_OP_CAN_CANCEL	Operation can be cancelled.

The RmiOpCanCancel enumeration is used in the following types:

- RmiResult
- RmiResultDataIncomplete

B4.6.38 RmiOpMemContig type

The RmiOpMemContig enumeration represents RMI operation memory contiguity property.

The RmiOpMemContig enumeration is a concrete type.

The width of the RmiOpMemContig enumeration is 1 bits.

See also:

- B4.3.2 *Stateful RMI operations*

The values of the RmiOpMemContig enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_OP_MEM_NON_CONTIG	Base of each block is aligned to the block size. Blocks are not required to be contiguous.
1	RMI_OP_MEM_CONTIG	A single contiguous region of memory, with the base aligned to the total size.

DRAFT

The RmiOpMemContig enumeration is used in the following types:

- RmiOpMemDonateReq

B4.6.39 RmiOpMemDonateReq type

The RmiOpMemDonateReq fieldset contains RMI operation memory donation requirements.

The RmiOpMemDonateReq fieldset is a concrete type.

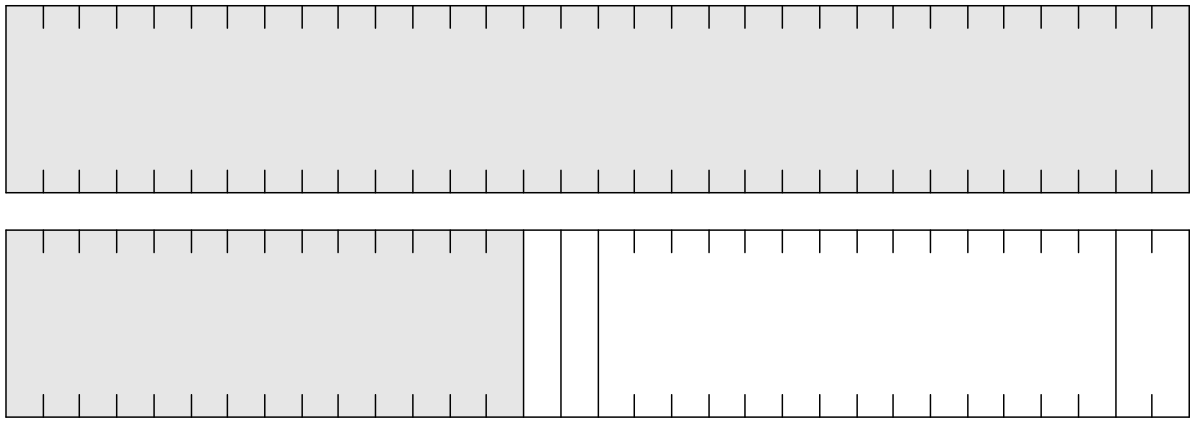
The width of the RmiOpMemDonateReq fieldset is 64 bits.

See also:

- B4.3.2 *Stateful RMI operations*

The fields of the RmiOpMemDonateReq fieldset are shown in the following diagram.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiOpMemDonateReq fieldset are shown in the following table.

Name	Bits	Description	Value
size	1:0	Size of each block	RmiAddrBlockSize
count	15:2	Number of blocks required	UInt14
contig	16	Whether blocks are required to be contiguous	RmiOpMemContig
state	17	Required initial state of memory	RmiOpMemState
	63:18	Reserved	SBZ

DRAFT

B4.6.40 RmiOpMemReq type

The RmiOpMemReq enumeration represents RMI operation memory transfer requirements.

The RmiOpMemReq enumeration is a concrete type.

The width of the RmiOpMemReq enumeration is 2 bits.

See also:

- B4.3.2 *Stateful RMI operations*

The values of the RmiOpMemReq enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_OP_MEM_REQ_NONE	RMI operation is not memory-transferring.
1	RMI_OP_MEM_REQ_DONATE	The RMI operation requires memory to be donated.
2	RMI_OP_MEM_REQ_RECLAIM	The RMI operation requires memory to be reclaimed.

Unused encodings for the RmiOpMemReq enumeration are reserved for use by future versions of this specification.

The RmiOpMemReq enumeration is used in the following types:

- RmiResult
- RmiResultDataIncomplete

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.41 RmiOpMemState type

The RmiOpMemState enumeration represents RMI operation memory state.

The RmiOpMemState enumeration is a concrete type.

The width of the RmiOpMemState enumeration is 1 bits.

The values of the RmiOpMemState enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_OP_MEM_DELEGATED	Initial state of memory is GRAN_DELEGATED.
1	RMI_OP_MEM_UNDELEGATED	Initial state of memory is GRAN_UNDELEGATED.

The RmiOpMemState enumeration is used in the following types:

- RmiAddrSetDesc
- RmiOpMemDonateReq
- RmiAddrRangeDesc64KB
- RmiAddrRangeDesc16KB
- RmiAddrRangeDesc
- RmiAddrRangeDesc4KB

B4.6.42 RmiPdevCategory type

The RmiPdevCategory enumeration represents PDEV category.

The RmiPdevCategory enumeration is a concrete type.

The width of the RmiPdevCategory enumeration is 2 bits.

The values of the RmiPdevCategory enumeration are shown in the following table.

DRAFT

Encoding	Name	Description
0	RMI_PDEV_ROOT_PORT	Root Port
1	RMI_PDEV_ENDPOINT_ACCEL_OFF_CHIP	Off-chip accelerator endpoint device
2	RMI_PDEV_ENDPOINT_ACCEL_ON_CHIP	On-chip accelerator endpoint device

Unused encodings for the RmiPdevCategory enumeration are reserved for use by future versions of this specification.

The RmiPdevCategory enumeration is used in the following types:

- RmiPdevFlags

B4.6.43 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.

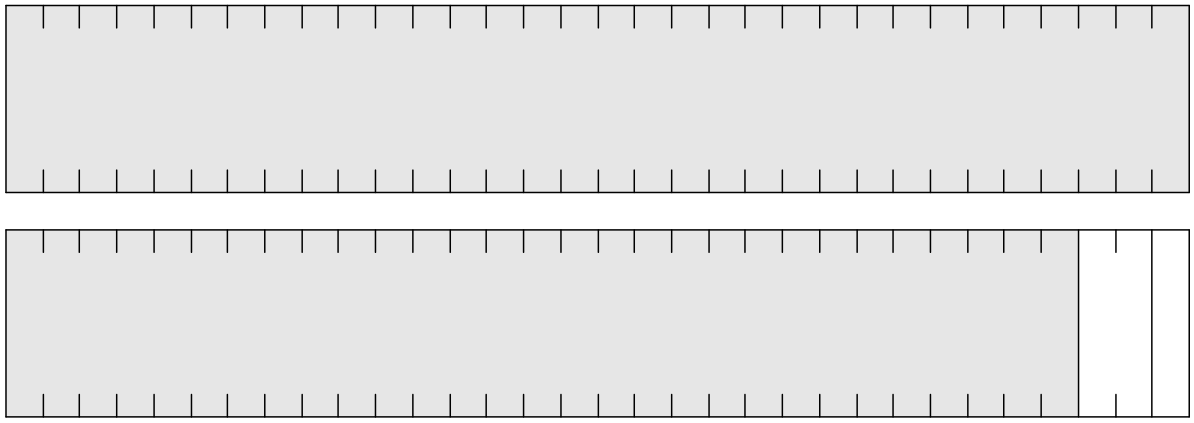
See also:

- A9.2.1 *Physical device attributes*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B3.120 *RmiPdevFlagsSupported function*

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
spdm	0	Whether communication with the device uses SPDM	RmiPdevSpdm
category	2:1	Device category	RmiPdevCategory
	63:3	Reserved	SBZ

DRAFT

The RmiPdevFlags fieldset is used in the following types:

- RmiPdevParams

B4.6.44 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
routing_id	`0x10`	Bits64	Routing identifier
id_index	`0x18`	UInt64	Device identity index
rid_base	`0x20`	Bits32	Base of requester ID range (inclusive). For PCIe category of devices, the value is in BDF format.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
rid_top	0x28	Bits32	Top of requester ID range (exclusive). For PCIe category of devices, the value is in BDF format.
hash_algo	0x30	RmiHashAlgorithm	Algorithm used to generate device digests
max_vdevs_order	0x38	UInt64	Order of the maximum number of VDEVs which can be created for this PDEV. The maximum number of VDEVs is computed as follows: `max_vdevs = (2^max_vdevs_order)- 1`

Unused bits of the RmiPdevParams structure SBZ.

B4.6.45 RmiPdevSpdm type

The RmiPdevSpdm enumeration represents whether communication with the device uses SPDM.

The RmiPdevSpdm enumeration is a concrete type.

The width of the RmiPdevSpdm enumeration is 1 bits.

See also:

- A9.1.2 *Device properties*

The values of the RmiPdevSpdm enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_SPDM_FALSE	Communication with the device does not use SPDM.
1	RMI_SPDM_TRUE	Communication with the device uses SPDM.

The RmiPdevSpdm enumeration is used in the following types:

- RmiPdevFlags

B4.6.46 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
3	RMI_PDEV_READY	Secure connection between the RMM and the device has been established.
4	RMI_PDEV_STOPPED	Secure connection between the RMM and the device has been terminated.
5	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.

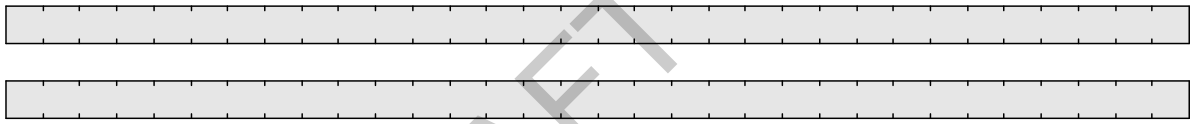
B4.6.47 RmiPdevStreamFlags type

The RmiPdevStreamFlags fieldset contains flags provided by the Host during PDEV stream creation.

The RmiPdevStreamFlags fieldset is a concrete type.

The width of the RmiPdevStreamFlags fieldset is 64 bits.

The fields of the RmiPdevStreamFlags fieldset are shown in the following diagram.



The fields of the RmiPdevStreamFlags fieldset are shown in the following table.

Name	Bits	Description	Value
	63:0	Reserved	SBZ

DRAFT

The RmiPdevStreamFlags fieldset is used in the following types:

- RmiPdevStreamParams

B4.6.48 RmiPdevStreamParams type

The RmiPdevStreamParams structure contains parameters provided by the Host during PDEV stream creation.

The RmiPdevStreamParams structure is a concrete type.

The width of the RmiPdevStreamParams structure is 4096 (`0x1000`) bytes.

The members of the RmiPdevStreamParams structure are shown in the following table.

Name	Byte offset	Type	Description
flags	`0x0`	RmiPdevStreamFlags	Flags
stream_type	`0x8`	RmiPdevStreamType	Stream type
pdev_1	`0x10`	Address	Address of first PDEV.
pdev_2	`0x18`	Address	Address of second PDEV
ide_sid	`0x20`	UInt64	IDE stream ID

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
num_addr_range	0x28	UInt64	Number of device address ranges
addr_range[16]	0x100	RmiAddrRange	CPU physical address range used to access the device in the system memory map

Unused bits of the RmiPdevStreamParams structure SBZ.

B4.6.49 RmiPdevStreamType type

The RmiPdevStreamType enumeration represents type of a PDEV stream.

The RmiPdevStreamType enumeration is a concrete type.

The width of the RmiPdevStreamType enumeration is 8 bits.

The values of the RmiPdevStreamType enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_PDEV_STREAM_NON_TEE	Non-TEE traffic.
1	RMI_PDEV_STREAM_NCOH	Non-coherent traffic between an upstream port and an endpoint device.
2	RMI_PDEV_STREAM_COH	Coherent traffic between an upstream port and an accelerator endpoint device.
3	RMI_PDEV_STREAM_NCOH_SYS	Non-coherent traffic to an endpoint device which is protected by system construction.
4	RMI_PDEV_STREAM_COH_SYS	Coherent traffic to an endpoint device which is protected by system construction.

DRAFT

Unused encodings for the RmiPdevStreamType enumeration are reserved for use by future versions of this specification.

The RmiPdevStreamType enumeration is used in the following types:

- RmiPdevStreamParams

B4.6.50 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

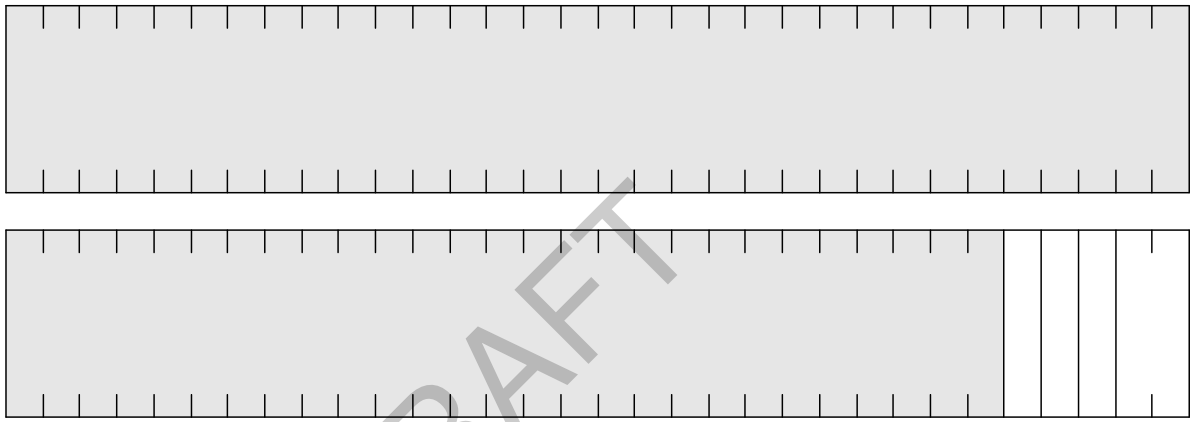
B4.6.51 RmiPsmmuFlags type

The RmiPsmmuFlags fieldset contains flags provided by the Host during PSMMU activation.

The RmiPsmmuFlags fieldset is a concrete type.

The width of the RmiPsmmuFlags fieldset is 64 bits.

The fields of the RmiPsmmuFlags fieldset are shown in the following diagram.



The fields of the RmiPsmmuFlags fieldset are shown in the following table.

Name	Bits	Description	Value
irq_cfg	1:0	IRQ configuration for GError, EventQ and PRIQ interrupts.	RmiIrqCfg
cmdq_sync_irq_wired	2	Whether command queue sync IRQ is wired	RmiFeature
ats	3	Whether to enable ATS	RmiFeature
pri	4	Whether to enable PRI	RmiFeature
	63:5	Reserved	SBZ

The RmiPsmmuFlags fieldset is used in the following types:

- RmiPsmmuParams

B4.6.52 RmiPsmmuInfo type

The RmiPsmmuInfo structure contains PSMMU info.

The RmiPsmmuInfo structure is a concrete type.

The width of the RmiPsmmuInfo structure is 4096 (`0x1000`) bytes.

The members of the RmiPsmmuInfo structure are shown in the following table.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
flags	0x0	RmiPsmmuInfoFlags	Flags
gerror_intr_num	0x8	UInt32	Gerror interrupt number for wired interrupt configuration.
eventq_intr_num	0x10	UInt32	EventQ interrupt number for wired interrupt configuration.
priq_intr_num	0x18	UInt32	PRIQ interrupt number for wired interrupt configuration.
cmdq_sync_intr_num	0x20	UInt32	CMD sync interrupt number for wired interrupt configuration.

Unused bits of the RmiPsmmuInfo structure SBZ.

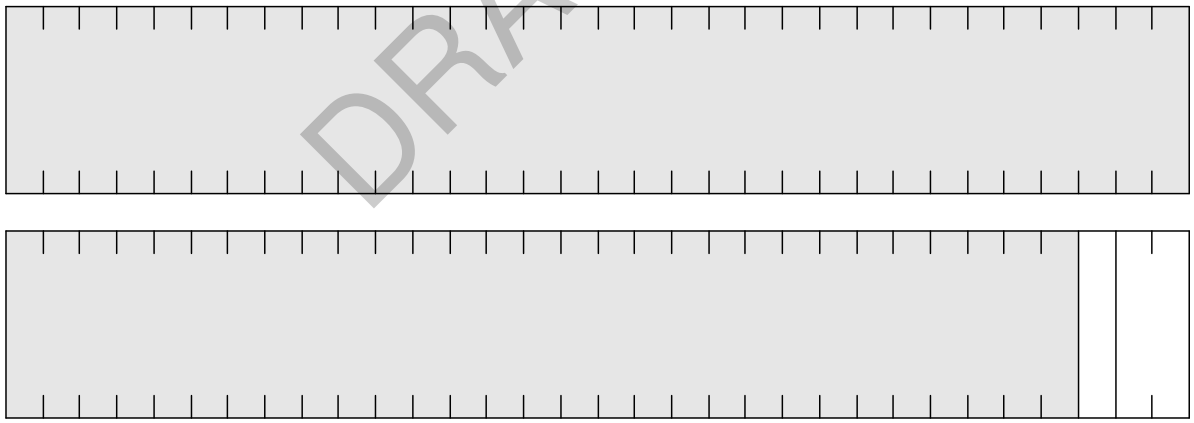
B4.6.53 RmiPsmmuInfoFlags type

The RmiPsmmuInfoFlags fieldset contains flags which describe the PSMMU configuration.

The RmiPsmmuInfoFlags fieldset is a concrete type.

The width of the RmiPsmmuInfoFlags fieldset is 64 bits.

The fields of the RmiPsmmuInfoFlags fieldset are shown in the following diagram.



The fields of the RmiPsmmuInfoFlags fieldset are shown in the following table.

Name	Bits	Description	Value
irq_cfg	1:0	IRQ configuration for GError, EventQ and PRIQ interrupts.	RmiIrqCfg
cmdq_sync_irq_wired	2	Whether command queue sync IRQ is wired	RmiFeature
	63:3	Reserved	SBZ

The RmiPsmmuInfoFlags fieldset is used in the following types:

- RmiPsmmuInfo

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.54 RmiPsmmuIrq type

The RmiPsmmuIrq enumeration represents identifies a single PSMMU IRQ.

The RmiPsmmuIrq enumeration is a concrete type.

The width of the RmiPsmmuIrq enumeration is 2 bits.

The values of the RmiPsmmuIrq enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_PSMMU_IRQ_GERROR	GERROR interrupt.
1	RMI_PSMMU_IRQ_EVENTQ	EVENTQ interrupt.
2	RMI_PSMMU_IRQ_PRIQ	PRIQ interrupt.
3	RMI_PSMMU_IRQ_CMDQ	CMDQ interrupt.

B4.6.55 RmiPsmmuIrqEvent type

The RmiPsmmuIrqEvent enumeration represents PSMMU irq event reported back to Host as part of the IRQ notification.

The RmiPsmmuIrqEvent enumeration is a concrete type.

The width of the RmiPsmmuIrqEvent enumeration is 2 bits.

The values of the RmiPsmmuIrqEvent enumeration are shown in the following table.

DRAFT

Encoding	Name	Description
0	RMI_PSMMU_IRQ_EVENT_NONE	No host action.
1	RMI_PSMMU_IRQ_EVENT_ERROR	SMMU fatal error.
2	RMI_PSMMU_IRQ_EVENT_PSMMU	SMMU event.
3	RMI_PSMMU_IRQ_EVENT_VSMMU	VSMMU event.

The RmiPsmmuIrqEvent enumeration is used in the following types:

- RmiPsmmuIrqResult

B4.6.56 RmiPsmmuIrqResult type

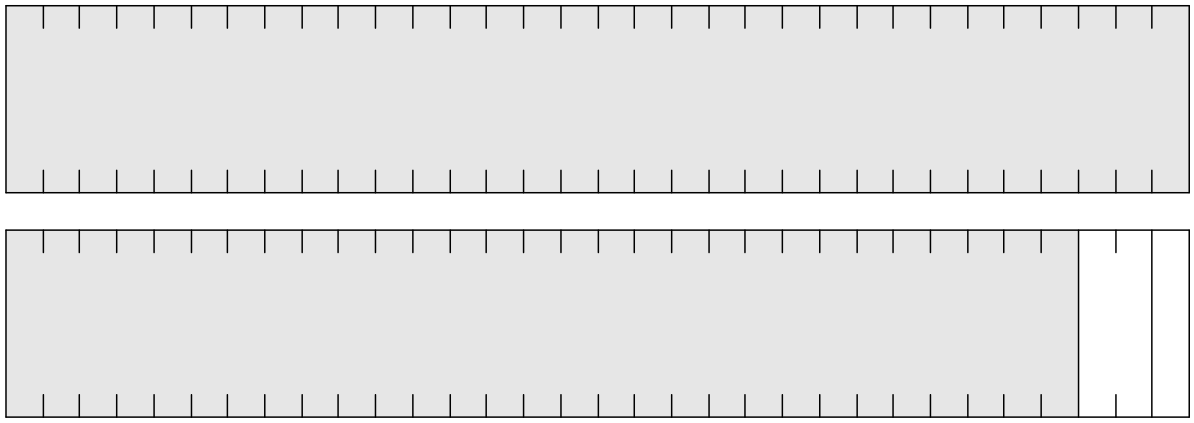
The RmiPsmmuIrqResult fieldset contains flags which describe the result of triaging a PSMMU IRQ.

The RmiPsmmuIrqResult fieldset is a concrete type.

The width of the RmiPsmmuIrqResult fieldset is 64 bits.

The fields of the RmiPsmmuIrqResult fieldset are shown in the following diagram.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiPsmmuIrqResult fieldset are shown in the following table.

Name	Bits	Description	Value
pending	0	Whether there are further PSMMU IRQs pending for processing	RmiBoolean
irq_event	2:1	PSMMU irq event reported back to the Host	RmiPsmmuIrqEvent
	63:3	Reserved	SBZ

DRAFT

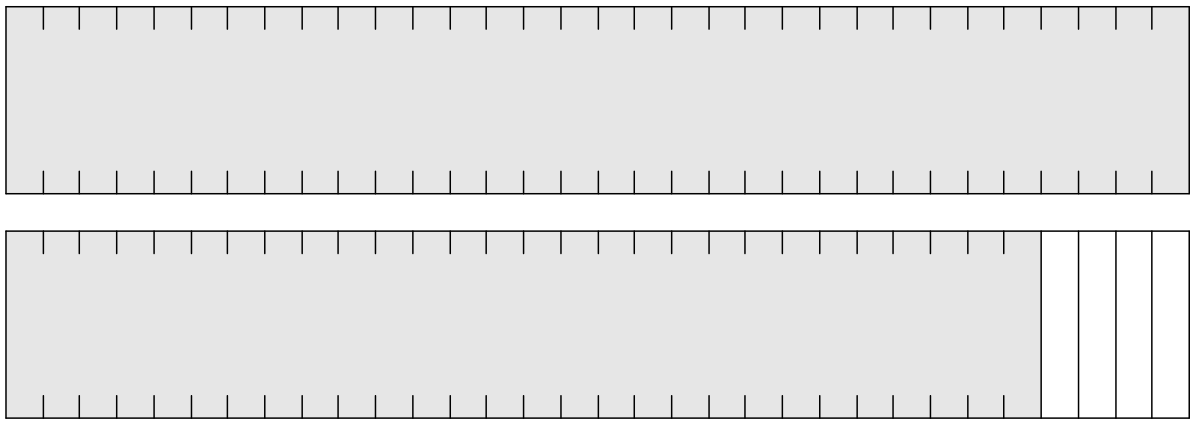
B4.6.57 RmiPsmmuIrqSet type

The RmiPsmmuIrqSet fieldset contains identifies a set of pending PSMMU IRQs.

The RmiPsmmuIrqSet fieldset is a concrete type.

The width of the RmiPsmmuIrqSet fieldset is 64 bits.

The fields of the RmiPsmmuIrqSet fieldset are shown in the following diagram.



The fields of the RmiPsmmuIrqSet fieldset are shown in the following table.

Name	Bits	Description	Value
gerror	0	Whether GERROR interrupt is pending	RmiBoolean

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Bits	Description	Value
eventq	1	Whether EVENTQ interrupt is pending	RmiBoolean
priq	2	Whether PRIQ interrupt is pending	RmiBoolean
cmdq_sync	3	Whether CMDQ-SYNC interrupt is pending	RmiBoolean
	63:4	Reserved	SBZ

B4.6.58 RmiPsmmuParams type

The RmiPsmmuParams structure contains PSMMU parameters.

The RmiPsmmuParams structure is a concrete type.

The width of the RmiPsmmuParams structure is 4096 (`0x1000`) bytes.

The members of the RmiPsmmuParams structure are shown in the following table.

Name	Byte offset	Type	Description
flags	`0x0`	RmiPsmmuFlags	Flags
gerr_addr	`0x8`	Address	Physical MSI address of the GERROR interrupt
gerr_data	`0x10`	Bits64	Physical MSI data of the GERROR interrupt
eventq_addr	`0x18`	Address	Physical MSI address of the EVENTQ interrupt
eventq_data	`0x20`	Bits64	Physical MSI data of the EVENTQ interrupt
priq_addr	`0x28`	Address	Physical MSI address of the PRIQ interrupt
priq_data	`0x30`	Bits64	Physical MSI data of the PRIQ interrupt

DRAFT

Unused bits of the RmiPsmmuParams structure SBZ.

B4.6.59 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
metadata_len	`0x808`	UInt64	Length of key metadata in bytes
algo	`0x810`	RmiSignatureAlgorithm	Signature algorithm

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Unused bits of the RmiPublicKeyParams structure SBZ.

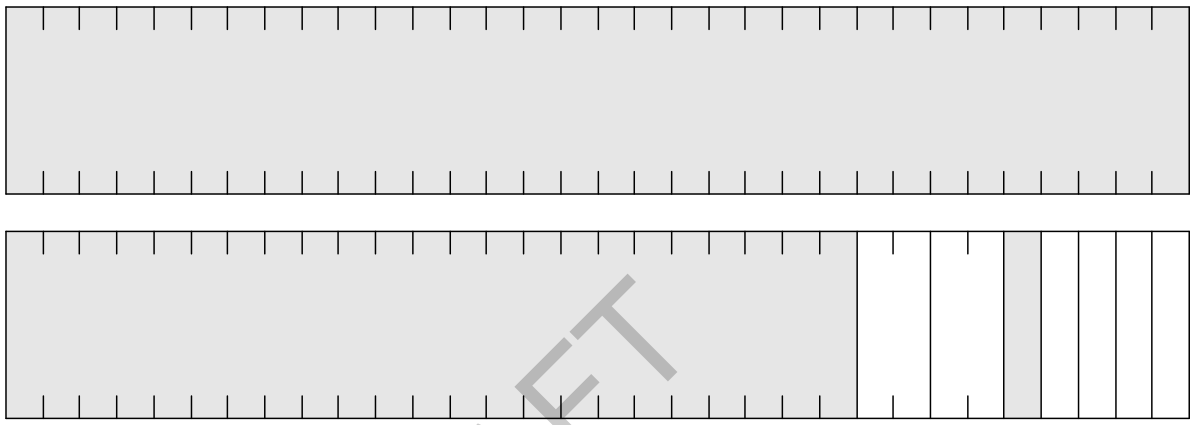
B4.6.60 RmiRealmFlags0 type

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
mec_policy	8:7	MEC policy	RmiMecPolicy
	63:9	Reserved	SBZ

The RmiRealmFlags0 fieldset is used in the following types:

- RmiRealmParams

B4.6.61 RmiRealmFlags1 type

The RmiRealmFlags1 fieldset contains flags provided by the Host during Realm creation, which are not reflected in Realm Initial Measurement.

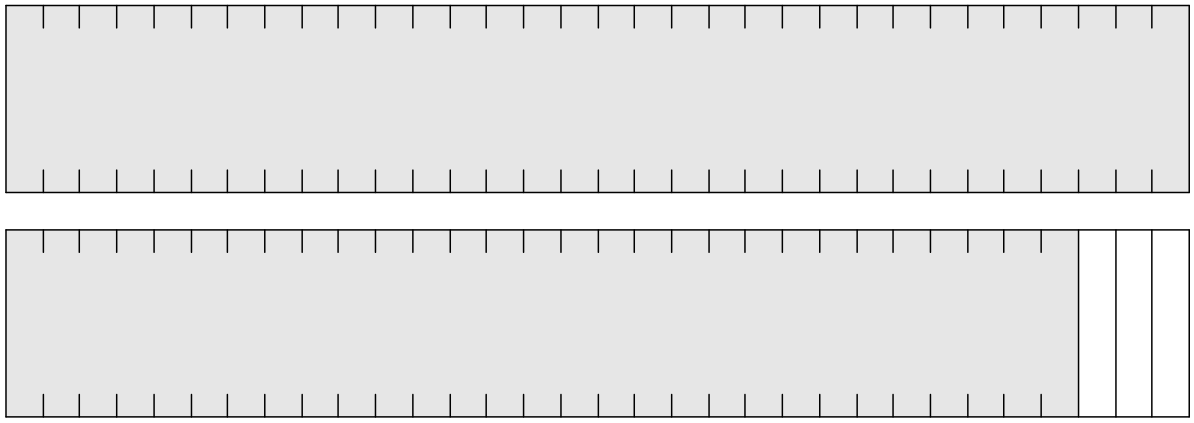
The RmiRealmFlags1 fieldset is a concrete type.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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
ats	2	Whether Address Translation Service is supported for devices assigned to the Realm	RmiFeature
	63:3	Reserved	SBZ

DRAFT

The RmiRealmFlags1 fieldset is used in the following types:

- RmiRealmParams

B4.6.62 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:

- A2.2.6 *Realm parameters*
- B4.5.37 *RMI_REALM_CREATE command*

The members of the RmiRealmParams structure are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 `VTCR_EL2.T0SZ` encoding.
sve_vl	0x10	UInt8	SVE vector length. The effective vector length requested is `(sve_vl + 1)*128`, similar to the value of `ZCR_ELx.LEN`.
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
ats_plane	0x440	UInt64	Index of Plane whose stage 2 permissions are observed by ATS requests from devices assigned to the Realm
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
aux_rtt_base[3]	0xf80	Address	Base address of auxiliary RTTs

Unused bits of the RmiRealmParams structure SBZ.

B4.6.63 RmiRecCreateFlags type

The RmiRecCreateFlags fieldset contains flags provided by the Host during REC creation.

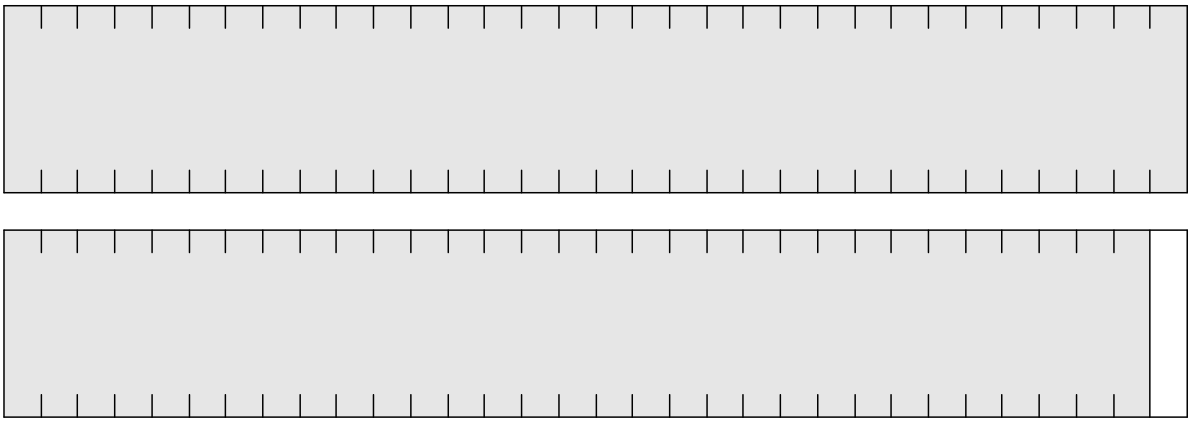
The RmiRecCreateFlags fieldset is a concrete type.

The width of the RmiRecCreateFlags fieldset is 64 bits.

The fields of the RmiRecCreateFlags fieldset are shown in the following diagram.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



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

DRAFT

B4.6.64 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*
- B4.5.42 *RMI_REC_ENTER command*
- B4.6.66 *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

Unused bits of the RmiRecEnter structure SBZ.

The RmiRecEnter structure is used in the following types:

- RmiRecRun

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

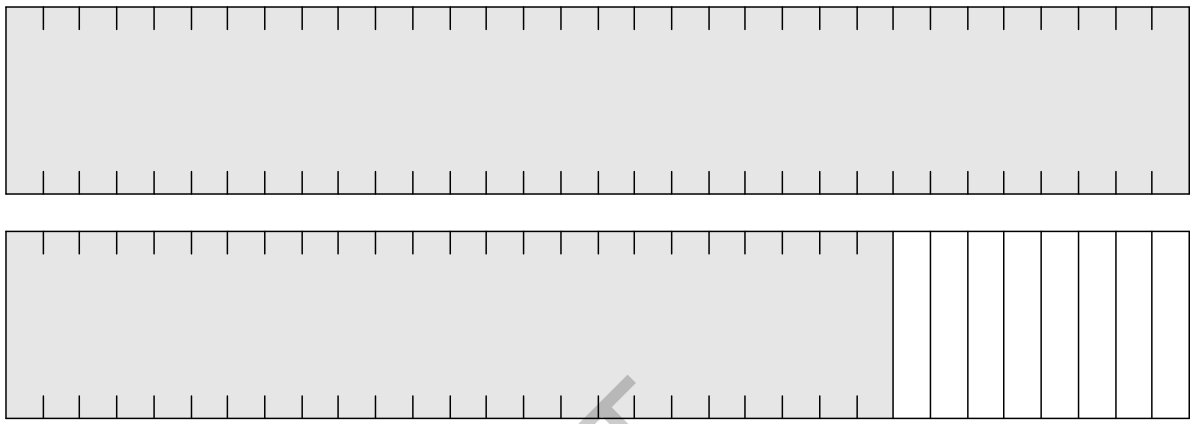
B4.6.65 RmiRecEnterFlags type

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 VDEV mapping validation request.	RmiResponse
force_p0	7	Whether to force control to return Plane 0	RmiForceP0
	63:8	Reserved	SBZ

DRAFT

The RmiRecEnterFlags fieldset is used in the following types:

- RmiRecEnter

B4.6.66 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:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- A4.3.1 *RmiRecExit object*
- B4.5.42 *RMI_REC_ENTER command*
- B4.6.64 *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
gprs[31]	0x200	Bits64	Registers
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_1	0x530	Bits64	Virtual device ID 1
vdev_id_2	0x538	Bits64	Virtual device ID 2
dev_mem_base	0x540	Bits64	Base IPA of target region for VDEV mapping validation
dev_mem_top	0x548	Bits64	Top IPA of target region for VDEV mapping validation
dev_mem_pa	0x550	Address	Base PA of device memory region
imm	0x600	Bits16	Host call immediate value
plane	0x608	UInt64	Plane index
pmu_ovf_status	0x700	RmiPmuOverflowStatus	PMU overflow status
vsmmu	0x710	Address	Virtual SMMU base IPA

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Unused bits of the RmiRecExit structure MBZ.

The RmiRecExit structure is used in the following types:

- RmiRecRun

B4.6.67 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_VALIDATE_MAPPING	REC exit due to VDEV mapping validation
10	RMI_EXIT_VSMMU_COMMAND	REC exit due to VSMMU command

DRAFT

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

B4.6.68 RmiRecMpidr type

The RmiRecMpidr fieldset contains MPIDR value which identifies a REC.

The RmiRecMpidr fieldset is a concrete type.

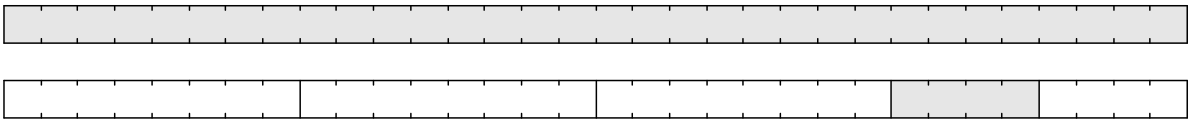
The width of the RmiRecMpidr fieldset is 64 bits.

See also:

- A2.4.3 *REC index and MPIDR value*
- B4.5.40 *RMI_REC_CREATE command*

The fields of the RmiRecMpidr fieldset are shown in the following diagram.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



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

B4.6.69 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 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
pc	`0x200`	Bits64	Program counter
gprs[8]	`0x300`	Bits64	General-purpose registers

Unused bits of the RmiRecParams structure SBZ.

B4.6.70 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*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.42 *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

B4.6.71 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

B4.6.72 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_RESPONSE_ACCEPT	Host accepted the Realm request.
1	RMI_RESPONSE_REJECT	Host rejected the Realm request.

The RmiResponse enumeration is used in the following types:

- RmiRecEnterFlags

B4.6.73 RmiResult type

The RmiResult fieldset contains a return code from an RMI command.

The RmiResult fieldset is a concrete type.

The width of the RmiResult fieldset is 64 bits.

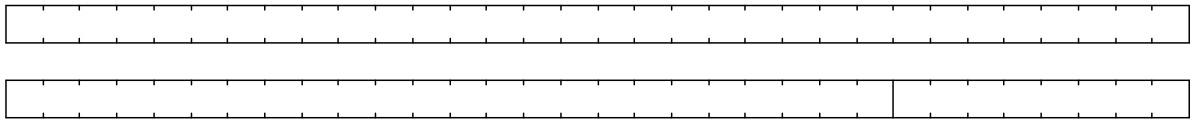
See also:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- Chapter B1 *Commands*

The fields of the RmiResult fieldset are shown in the following diagram.



The fields of the RmiResult fieldset are shown in the following table.

Name	Bits	Description	Value
status	7:0	Status of the command	RmiStatusCode
data	63:8	Conditional	See below

Encodings for field data:

Name	Condition	Fieldset
incomplete	`status == RMI_INCOMPLETE`	RmiResultDataIncomplete
level	`status IN { RMI_ERROR_DPT, RMI_ERROR_RTT, RMI_ERROR_RTT_AUX, RMI_ERROR_PSMMU_ST }`	RmiResultDataLevel
null	`!status IN { RMI_ERROR_DPT, RMI_ERROR_RTT, RMI_ERROR_RTT_AUX, RMI_ERROR_PSMMU_ST, RMI_INCOMPLETE }`	RmiResultDataNull

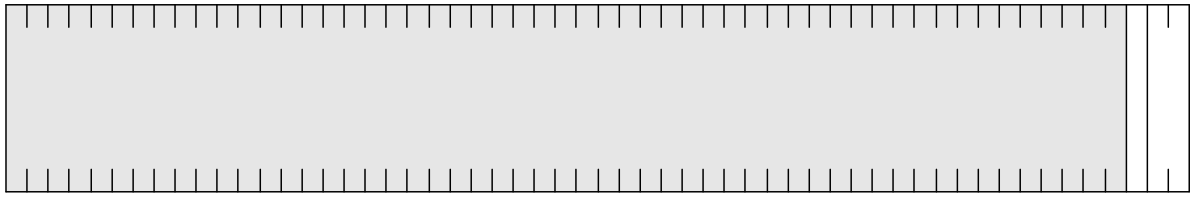
B4.6.74 RmiResultDataIncomplete type

The RmiResultDataIncomplete fieldset contains encoding for additional data from an RMI command which returned RMI_INCOMPLETE.

The RmiResultDataIncomplete fieldset is a concrete type.

The width of the RmiResultDataIncomplete fieldset is 56 bits.

The fields of the RmiResultDataIncomplete fieldset are shown in the following diagram.



The fields of the RmiResultDataIncomplete fieldset are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Bits	Description	Value
mem	1:0	Memory transfer requirements	RmiOpMemReq
cancel	2	Whether operation can be cancelled	RmiOpCanCancel
	55:3	Reserved	MBZ

The RmiResultDataIncomplete fieldset is used in the following types:

- RmiResult

B4.6.75 RmiResultDataLevel type

The RmiResultDataLevel fieldset contains encoding for additional data from an RMI command which indicates a level.

The RmiResultDataLevel fieldset is a concrete type.

The width of the RmiResultDataLevel fieldset is 56 bits.

The fields of the RmiResultDataLevel fieldset are shown in the following diagram.

The fields of the RmiResultDataLevel fieldset are shown in the following table.

DRAFT

Name	Bits	Description	Value
level	7:0	Level at which access to a table or other data structure terminated	UInt8
	55:8	Reserved	MBZ

The RmiResultDataLevel fieldset is used in the following types:

- RmiResult

B4.6.76 RmiResultDataNull type

The RmiResultDataNull fieldset contains encoding for null additional data from an RMI command.

The RmiResultDataNull fieldset is a concrete type.

The width of the RmiResultDataNull fieldset is 56 bits.

The fields of the RmiResultDataNull fieldset are shown in the following diagram.

The fields of the RmiResultDataNull fieldset are shown in the following table.

Name	Bits	Description	Value
	55:0	Reserved	MBZ

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The RmiResultDataNull fieldset is used in the following types:

- RmiResult

B4.6.77 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_RIPAS_EMPTY	Address where no Realm resources are mapped.
1	RMI_RIPAS_RAM	Address where private code or data owned by the Realm is mapped.
2	RMI_RIPAS_DESTROYED	Address which is inaccessible to the Realm due to an action taken by the Host.
3	RMI_RIPAS_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

B4.6.78 RmiRmmConfig type

The RmiRmmConfig structure contains global configuration of the RMM.

The RmiRmmConfig structure is a concrete type.

The width of the RmiRmmConfig structure is 4096 (`0x1000`) bytes.

The members of the RmiRmmConfig structure are shown in the following table.

Name	Byte offset	Type	Description
tracking_region_size	`0x0`	UInt8	Tracking region size The encoding of this value depends on the RMI Granule size, as follows: RMI Granule size = 4KB • 0: Tracking region size = 1GB RMI Granule size = 16KB • 0: Tracking region size = 32MB • 1: Tracking region size = 64GB RMI Granule size = 64KB • 0: Tracking region size = 512MB • 1: Tracking region size = 4TB All other values are reserved.
rmi_granule_size	`0x8`	RmiGranuleSize	RMI Granule size

Unused bits of the RmiRmmConfig structure SBZ.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.79 RmiRmmState type

The RmiRmmState enumeration represents RMM state.

The RmiRmmState enumeration is a concrete type.

The width of the RmiRmmState enumeration is 8 bits.

The values of the RmiRmmState enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_RMM_STATE_INIT	Initial state of the RMM.
1	RMI_RMM_STATE_ACTIVE	RMM is active.

Unused encodings for the RmiRmmState enumeration are reserved for use by future versions of this specification.

B4.6.80 RmiRttAddrType type

The RmiRttAddrType enumeration represents meaning of an address value which is the input or output of an RTT command.

The RmiRttAddrType enumeration is a concrete type.

The width of the RmiRttAddrType enumeration is 2 bits.

The values of the RmiRttAddrType enumeration are shown in the following table.

DRAFT

Encoding	Name	Description
0	RMI_ADDR_TYPE_NONE	Address value is ignored
1	RMI_ADDR_TYPE_SINGLE	Address value is a single RMI Address Range Descriptor
2	RMI_ADDR_TYPE_LIST	Address value is the address of a Granule that holds a list of RMI Address Range Descriptors

Unused encodings for the RmiRttAddrType enumeration are reserved for use by future versions of this specification.

The RmiRttAddrType enumeration is used in the following types:

- RmiRttUnprotMapFlags
- RmiRttUnmapFlags
- RmiRttProtMapFlags

B4.6.81 RmiRttAuxBlock type

The RmiRttAuxBlock enumeration represents control behaviour of auxiliary RTT mapping operation when target IPA range is a subset of a block mapping in the auxiliary RTT tree.

The RmiRttAuxBlock enumeration is a concrete type.

The width of the RmiRttAuxBlock enumeration is 1 bits.

The values of the RmiRttAuxBlock enumeration are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
0	RMI_RTT_AUX_BLOCK_NO_CREATE	Do not create a block mapping in the auxiliary RTT tree
1	RMI_RTT_AUX_BLOCK_CREATE	Create a block mapping in the auxiliary RTT tree

The RmiRttAuxBlock enumeration is used in the following types:

- RmiRttAuxMapFlags

B4.6.82 RmiRttAuxInvalidPri type

The RmiRttAuxInvalidPri enumeration represents control behaviour of auxiliary RTT mapping operation when primary RTTE is invalid.

The RmiRttAuxInvalidPri enumeration is a concrete type.

The width of the RmiRttAuxInvalidPri enumeration is 1 bits.

The values of the RmiRttAuxInvalidPri enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_RTT_AUX_INVALID_PRI_STOP	Stop the operation and return an error to the caller
1	RMI_RTT_AUX_INVALID_PRI_CONTINUE	Set auxiliary RTTE to invalid and continue the operation

DRAFT

The RmiRttAuxInvalidPri enumeration is used in the following types:

- RmiRttAuxMapFlags

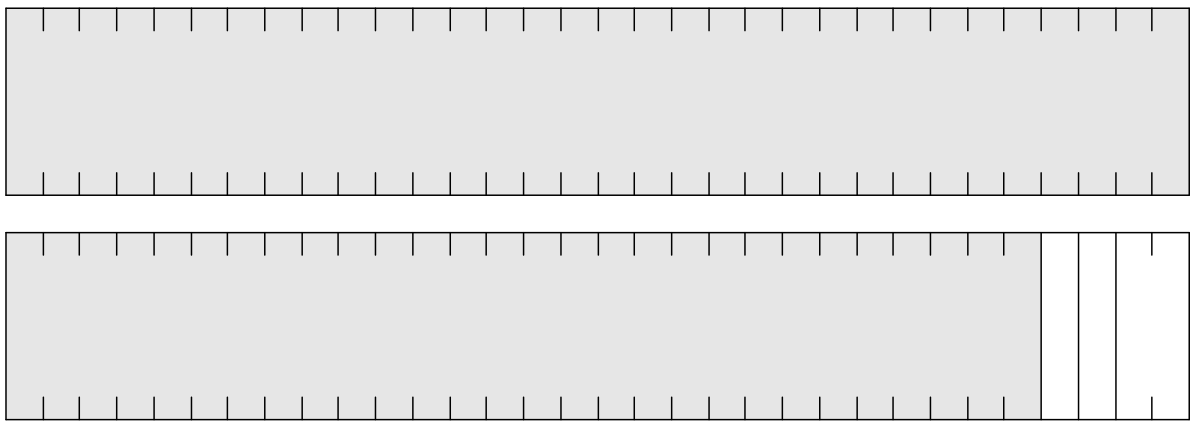
B4.6.83 RmiRttAuxMapFlags type

The RmiRttAuxMapFlags fieldset contains flags provided by the Host to commands which create mappings in an auxiliary RTT tree.

The RmiRttAuxMapFlags fieldset is a concrete type.

The width of the RmiRttAuxMapFlags fieldset is 64 bits.

The fields of the RmiRttAuxMapFlags fieldset are shown in the following diagram.



Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The fields of the RmiRttAuxMapFlags fieldset are shown in the following table.

Name	Bits	Description	Value
tree_index	1:0	Index of auxiliary RTT tree	UInt2
block	2	Control behaviour when target IPA range is a subset of a block mapping in the auxiliary RTT tree	RmiRttAuxBlock
invalid_pri	3	Control behaviour when primary RTTE is invalid	RmiRttAuxInvalidPri
	63:4	Reserved	SBZ

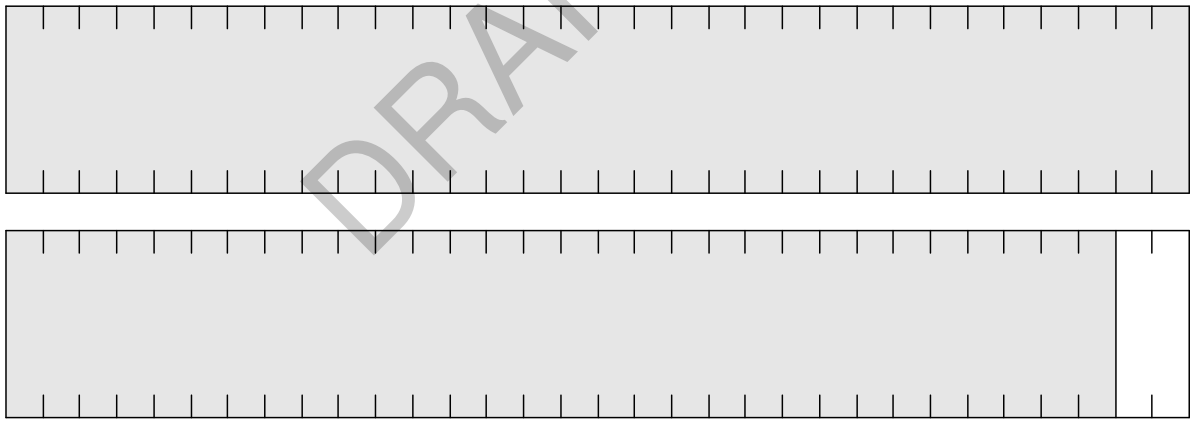
B4.6.84 RmiRttAuxUnmapFlags type

The RmiRttAuxUnmapFlags fieldset contains flags provided by the Host to commands which remove mappings in an auxiliary RTT tree.

The RmiRttAuxUnmapFlags fieldset is a concrete type.

The width of the RmiRttAuxUnmapFlags fieldset is 64 bits.

The fields of the RmiRttAuxUnmapFlags fieldset are shown in the following diagram.



The fields of the RmiRttAuxUnmapFlags fieldset are shown in the following table.

Name	Bits	Description	Value
tree_index	1:0	Index of auxiliary RTT tree	UInt2
	63:2	Reserved	SBZ

B4.6.85 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
0	RMI_RTTE_VOID	This RTTE is not associated with any Granule.
1	RMI_RTTE_DATA	The output address of this RTTE points to: • 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_RTTE_TABLE	The output address of this RTTE points to the next-level RTT.
3	RMI_RTTE_NARCH_DEV	The output address of this RTTE points to an GRAN_DEV Granule.
4	RMI_RTTE_AUX_DESTROYED	An auxiliary RTT was destroyed.
5	RMI_RTTE_ARCH_DEV	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.

B4.6.86 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.

DRAFT

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:

- RmiFeatureRegister3

B4.6.87 RmiRttProtMapFlags type

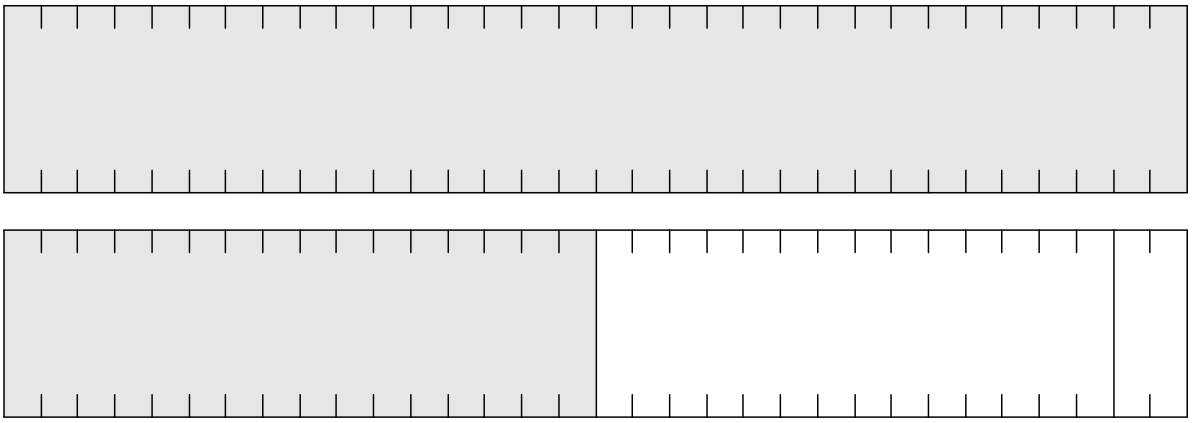
The RmiRttProtMapFlags fieldset contains flags provided by the Host to commands which create mappings in Protected IPA space.

The RmiRttProtMapFlags fieldset is a concrete type.

The width of the RmiRttProtMapFlags fieldset is 64 bits.

The fields of the RmiRttProtMapFlags fieldset are shown in the following diagram.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiRttProtMapFlags fieldset are shown in the following table.

Name	Bits	Description	Value
oaddr_type	1:0	Type of oaddr input value	RmiRttAddrType
list_count	15:2	Number of valid entries in the oaddr list. If oaddr_type != RMI_ADDR_TYPE_LIST then this field is ignored.	UInt14
	63:16	Reserved	SBZ

DRAFT

B4.6.88 RmiRttS2APBase type

The RmiRttS2APBase enumeration represents S2AP base value.

The RmiRttS2APBase enumeration is a concrete type.

The width of the RmiRttS2APBase enumeration is 4 bits.

The values of the RmiRttS2APBase enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_S2AP_NO_ACCESS	NoAccess
1	RMI_S2AP_RO	RO
2	RMI_S2AP_WO	WO
3	RMI_S2AP_RW	RW
4	RMI_S2AP_RW_PUX	RW+puX

Unused encodings for the RmiRttS2APBase enumeration are reserved for use by future versions of this specification.

B4.6.89 RmiRttS2APEncoding type

The RmiRttS2APEncoding enumeration represents S2AP encoding.

The RmiRttS2APEncoding enumeration is a concrete type.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The width of the RmiRttS2APEncoding enumeration is 1 bits.

See also:

- A3.12 *Support for Stage 2 Access Permissions indirect encoding*

The values of the RmiRttS2APEncoding enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_S2AP_DIRECT	S2AP is encoded directly in the RTT entry.
1	RMI_S2AP_INDIRECT	RTT entry includes indices which indirectly specify the S2AP.

The RmiRttS2APEncoding enumeration is used in the following types:

- RmiRealmFlags1

B4.6.90 RmiRttUnmapFlags type

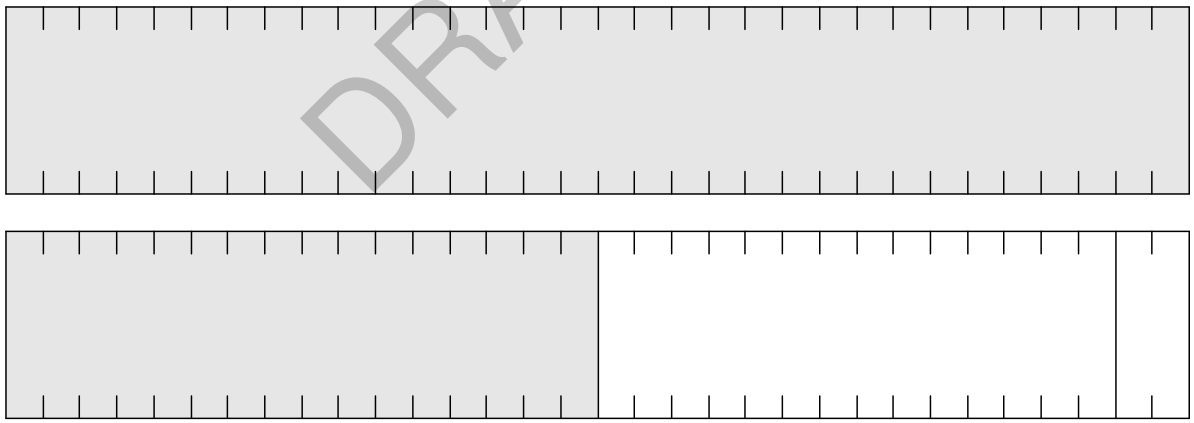
The RmiRttUnmapFlags fieldset contains flags provided by the Host to commands which remove mappings from IPA space.

The RmiRttUnmapFlags fieldset is a concrete type.

The width of the RmiRttUnmapFlags fieldset is 64 bits.

The fields of the RmiRttUnmapFlags fieldset are shown in the following diagram.

DRAFT



The fields of the RmiRttUnmapFlags fieldset are shown in the following table.

Name	Bits	Description	Value
oaddr_type	1:0	Type of oaddr input value	RmiRttAddrType
list_count	15:2	Number of valid entries in the oaddr list. If oaddr_type != RMI_ADDR_TYPE_LIST then this field is ignored.	UInt14
	63:16	Reserved	SBZ

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

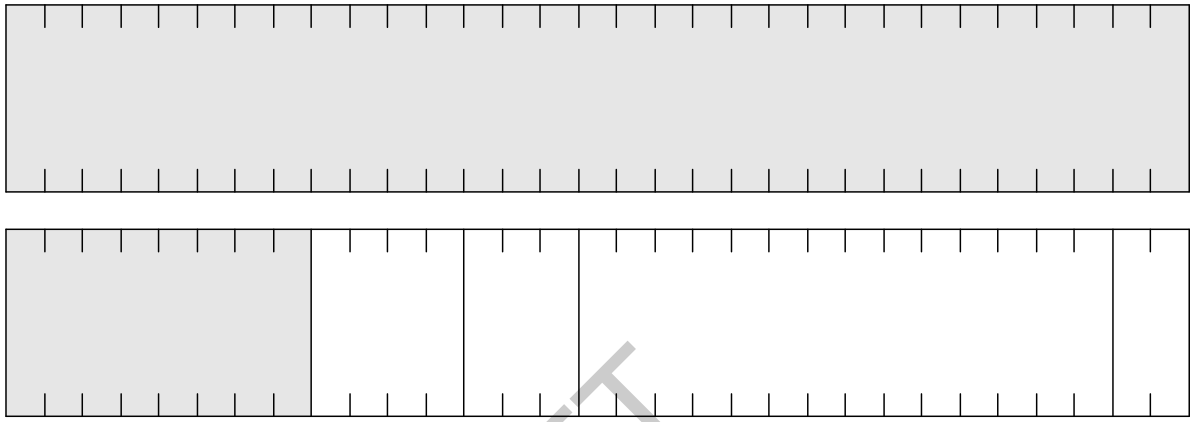
B4.6.91 RmiRttUnprotMapFlags type

The RmiRttUnprotMapFlags fieldset contains flags provided by the Host to commands which create mappings in Unprotected IPA space.

The RmiRttUnprotMapFlags fieldset is a concrete type.

The width of the RmiRttUnprotMapFlags fieldset is 64 bits.

The fields of the RmiRttUnprotMapFlags fieldset are shown in the following diagram.



The fields of the RmiRttUnprotMapFlags fieldset are shown in the following table.

Name	Bits	Description	Value
oaddr_type	1:0	Type of oaddr input value	RmiRttAddrType
list_count	15:2	Number of valid entries in the oaddr list. If oaddr_type != RMI_ADDR_TYPE_LIST then this field is ignored.	UInt14
memattr	18:16	Memory attributes. This field is encoded as for `MemAttr[2:0]` in a translation table page or block descriptor.	Bits3
s2ap	22:19	Stage 2 access permissions. If the Realm uses S2AP direct encoding, this is encoded as for `AP` in a translation table page or block descriptor. If the Realm uses S2AP indirect encoding, this is encoded as for `PIIndex` in a translation table page or block descriptor.	Bits4
	61:23	Reserved	SBZ

B4.6.92 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
0	RMI_SIG_RSASSA_3072	SSA-3072 (*RSA Cryptography Specifications Version 2.2* [26])
1	RMI_SIG_ECDSA_P256	ECDSA-P256 (*Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)* [27])
2	RMI_SIG_ECDSA_P384	ECDSA-P384 (*Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)* [27])

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

B4.6.93 RmiStatusCode type

The RmiStatusCode enumeration represents the status of an RMI command.

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.

DRAFT

Encoding	Name	Description
0	RMI_SUCCESS	Command completed successfully, leaving no intermediate state
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
8	RMI_ERROR_PSMMU_ST	An PSMMU Stream Table walk terminated before reaching the target level, or reached an entry with an unexpected value
9	RMI_ERROR_DPT	A DPT walk terminated before reaching the target level, or reached an entry with an unexpected value

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
10	RMI_BUSY	The command failed to make progress, for an IMPLEMENTATION DEFINED reason. The reason for the lack of progress may be temporary, and may be resolved without requiring any action to be taken by the Host. The command did not result in any changes of system state.
11	RMI_ERROR_GLOBAL	An attribute of RMM global state does not match the expected value
12	RMI_ERROR_TRACKING	The state of a tracking region does not match the expected value
13	RMI_INCOMPLETE	The command initiated a state transition but did not complete, leaving an object in an intermediate state. The target object cannot be the subject of any other RMI command, until this one has been completed.
14	RMI_BLOCKED	The command failed to make progress due to a target object being in an intermediate state. An incomplete operation must be completed before this one can proceed. The command did not result in any changes of system state.
15	RMI_ERROR_GPT	A GPT walk terminated before reaching the target level, or reached an entry with an unexpected value
16	RMI_ERROR_GRANULE	An attribute of a Granule does not match the expected 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:

- RmiResult

DRAFT

B4.6.94 RmiTrackingRegionState type

The RmiTrackingRegionState enumeration represents tracking region state.

The RmiTrackingRegionState enumeration is a concrete type.

The width of the RmiTrackingRegionState enumeration is 3 bits.

The values of the RmiTrackingRegionState enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_TRACKING_RESERVED	Region is reserved for use by the platform.
1	RMI_TRACKING_NONE	Region is not tracked.
2	RMI_TRACKING_FINE	Region is tracked at the granularity of the RMI Granule size.
3	RMI_TRACKING_COARSE	Region is tracked at the granularity of the Tracking Region size.

Unused encodings for the RmiTrackingRegionState enumeration are reserved for use by future versions of this specification.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.95 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

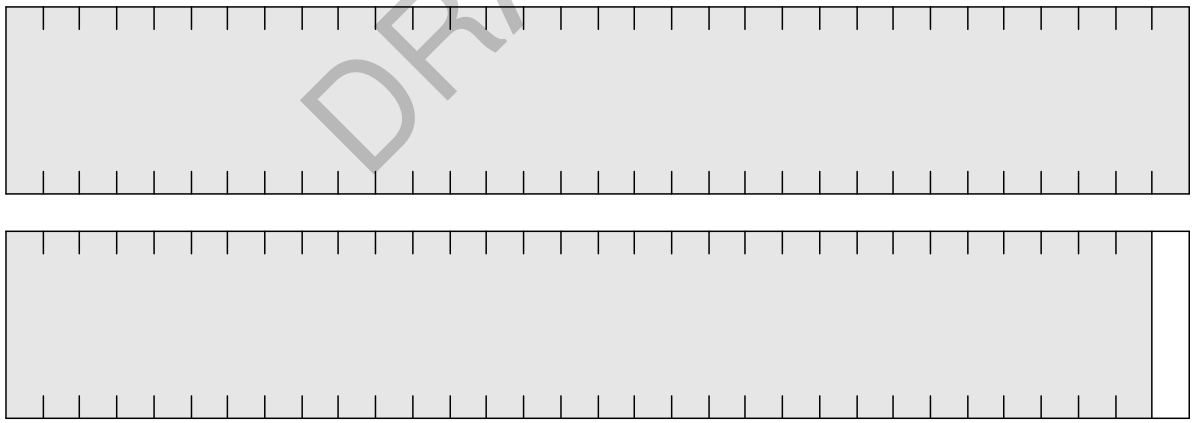
B4.6.96 RmiVdevFlags type

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.97 RmiVdevMeasureFlags type

The RmiVdevMeasureFlags fieldset contains flags which describe properties of device measurements.

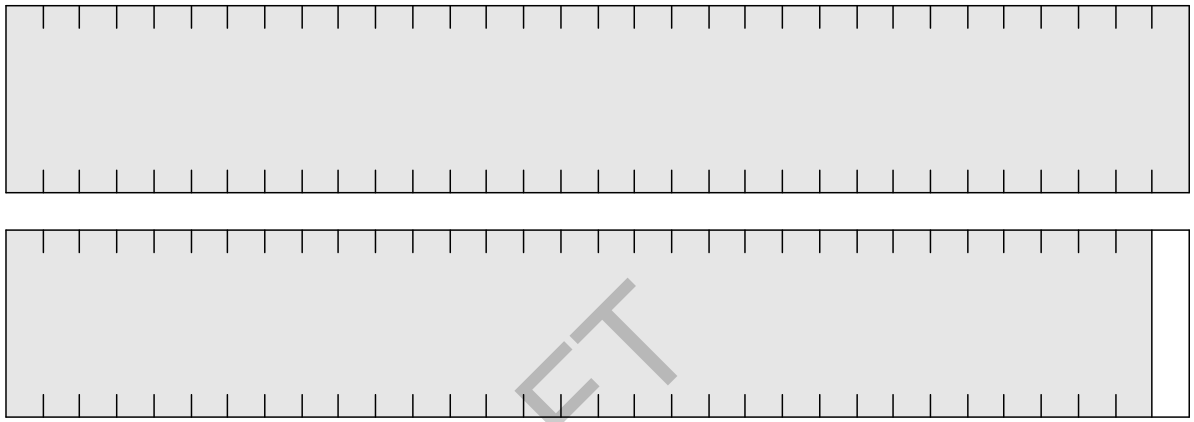
The RmiVdevMeasureFlags fieldset is a concrete type.

The width of the RmiVdevMeasureFlags fieldset is 64 bits.

See also:

- A9.5.5 *Device measurement retrieval*

The fields of the RmiVdevMeasureFlags fieldset are shown in the following diagram.



The fields of the RmiVdevMeasureFlags fieldset are shown in the following table.

Name	Bits	Description	Value
raw	0	Whether the return value is a raw bitstream.	RmiVdevMeasureRaw
	63:1	Reserved	SBZ

The RmiVdevMeasureFlags fieldset is used in the following types:

- RmiVdevMeasureParams

B4.6.98 RmiVdevMeasureParams type

The RmiVdevMeasureParams structure contains device measurement parameters.

The RmiVdevMeasureParams structure is a concrete type.

The width of the RmiVdevMeasureParams structure is 4096 (`0x1000`) bytes.

See also:

- A9.5.5 *Device measurement retrieval*

The members of the RmiVdevMeasureParams structure are shown in the following table.

Name	Byte offset	Type	Description
flags	`0x0`	RmiVdevMeasureFlags	Attestation type

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
nonce	0x100	Bits256	Nonce value used in requests for signed measurements.

Unused bits of the RmiVdevMeasureParams structure MBZ.

B4.6.99 RmiVdevMeasureRaw type

The RmiVdevMeasureRaw enumeration represents whether a device measurement is a raw bitstream.

The RmiVdevMeasureRaw enumeration is a concrete type.

The width of the RmiVdevMeasureRaw enumeration is 1 bits.

See also:

- A9.5.5 *Device measurement retrieval*

The values of the RmiVdevMeasureRaw enumeration are shown in the following table.

Encoding	Name	Description
0	RMI_VDEV_MEASURE_NOT_RAW	Returned value is measurement hash
1	RMI_VDEV_MEASURE_RAW	Returned value is a raw bitstream

The RmiVdevMeasureRaw enumeration is used in the following types:

- RmiVdevMeasureFlags

B4.6.100 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
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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
num_addr_range	0x30	UInt64	Number of device address ranges.
addr_range[8]	0x200	RmiAddrRange	CPU physical address range used to access the device in the system memory map These ranges include both coherent and non-coherent device memory.

Unused bits of the RmiVdevParams structure SBZ.

B4.6.101 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 interface.
1	RMI_VDEV_UNLOCKED	Device interface is unlocked.
2	RMI_VDEV_LOCKED	Device interface is locked.
3	RMI_VDEV_STARTED	Device interface is started.
4	RMI_VDEV_ERROR	Device interface has reported a fatal error.
5	RMI_VDEV_KEY_REFRESH	Waiting for key refresh to be performed on all streams associated with the device interface.
6	RMI_VDEV_KEY_PURGE	Waiting for purge of inactive keys to be performed on all streams associated with the device interface.

DRAFT

Unused encodings for the RmiVdevState enumeration are reserved for use by future versions of this specification.

B4.6.102 RmiVsmmuCmdFlags type

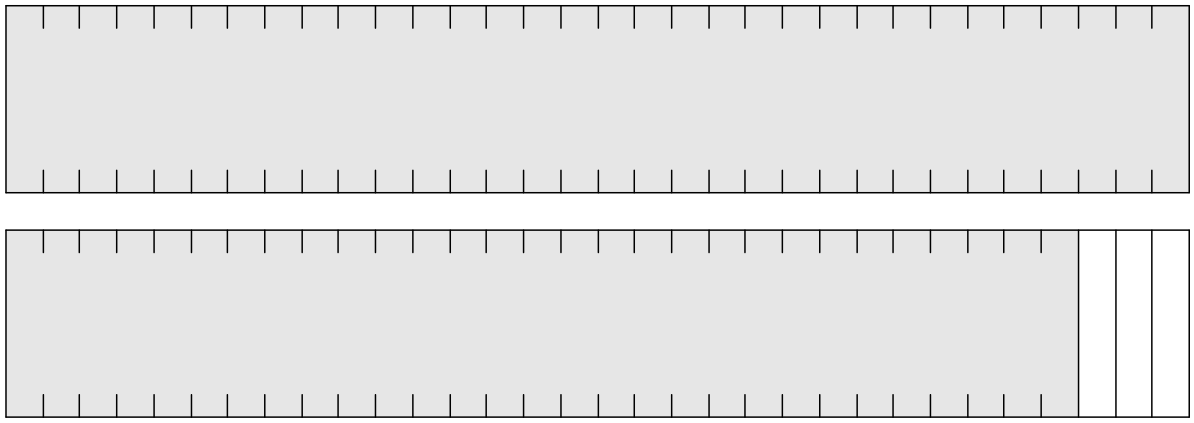
The RmiVsmmuCmdFlags fieldset contains flags which describe the result of triaging a VSMMU command.

The RmiVsmmuCmdFlags fieldset is a concrete type.

The width of the RmiVsmmuCmdFlags fieldset is 64 bits.

The fields of the RmiVsmmuCmdFlags fieldset are shown in the following diagram.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RmiVsmmuCmdFlags fieldset are shown in the following table.

Name	Bits	Description	Value
pending	0	Whether a further VSMMU command is pending	RmiBoolean
cmd_complete	1	Whether VSMMU command requires further processing by the RMM	RmiBoolean
irq	2	Whether VSMMU event requires an IRQ to be injected into the Realm	RmiBoolean
	63:3	Reserved	SBZ

DRAFT

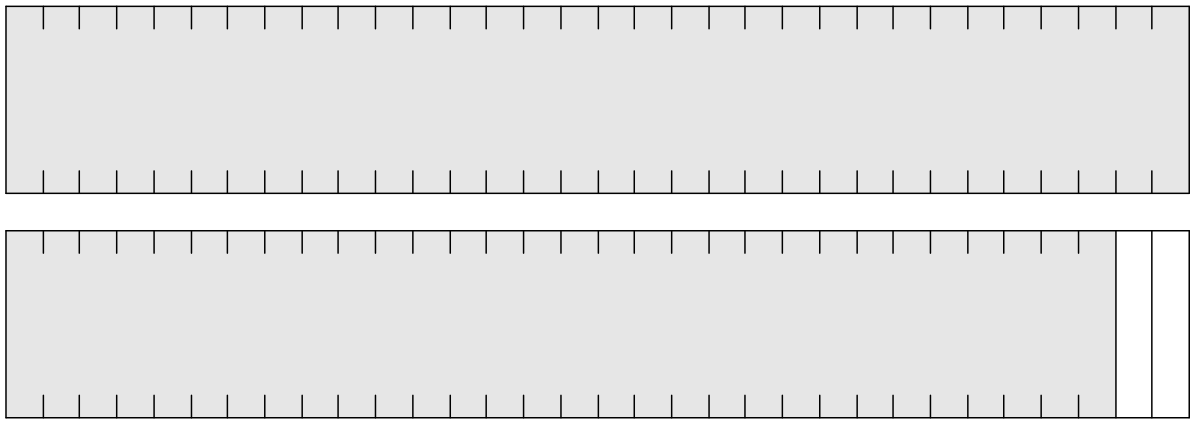
B4.6.103 RmiVsmmuEventFlags type

The RmiVsmmuEventFlags fieldset contains flags which describe the result of triaging a VSMMU event.

The RmiVsmmuEventFlags fieldset is a concrete type.

The width of the RmiVsmmuEventFlags fieldset is 64 bits.

The fields of the RmiVsmmuEventFlags fieldset are shown in the following diagram.



The fields of the RmiVsmmuEventFlags fieldset are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Bits	Description	Value
irq	0	Whether VSMMU event requires an IRQ to be injected into the Realm	RmiBoolean
abort	1	Whether RMM access to VSMMU queue failed due to Data Abort	RmiBoolean
	63:2	Reserved	SBZ

B4.6.104 RmiVsmmuFeatures type

The RmiVsmmuFeatures structure contains VSMMU features supported by the RMM.

The RmiVsmmuFeatures structure is a concrete type.

The width of the RmiVsmmuFeatures structure is 256 (`0x100`) bytes.

The members of the RmiVsmmuFeatures structure are shown in the following table.

Name	Byte offset	Type	Description
aidr	`0x0`	Bits64	SMMU_AIDR register value
idr[7]	`0x8`	Bits64	SMMU_IDR register values

Unused bits of the RmiVsmmuFeatures structure SBZ.

DRAFT

B4.6.105 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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B4.6.106 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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter B5
Realm Services Interface

DRAFT

This chapter defines the interface used by Realm software to request services from the RMM.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.1 RSI version

R_{QKLGZ} This specification defines version 1.1 of the Realm Services Interface.

See also:

- Chapter B2 *Interface versioning*
- B5.4.21 *RSI_VERSION command*

B5.2 Programming models for RSI operations

D_{MQDGF} An *RSI operation* is an operation which is performed by execution of RSI commands.

See also:

- B5.2 *Programming models for RSI operations*

B5.2.1 Properties of RSI operations

D_{FFJRY} A *range RSI operation* is an operation which modifies the state of a set of objects, which are identified by a contiguous range of addresses.

D_{GFMHC} A *non-range RSI operation* is an operation which modifies a single object or a set of objects, each of which is identified separately.

D_{TVFKK} A *long-running RSI operation* is an operation which may require multiple RSI calls in order to complete the operation.

Specifying that an RSI operation is long-running allows the implementation to guarantee that execution of the command will not cause delivery of interrupts to be delayed by more than an IMPLEMENTATION DEFINED upper bound.

D_{TDQKB} A *non-long-running RSI operation* is an operation which does not require multiple RSI calls in order to complete the operation.

I_{DZTZF} A range RSI operation is a long-running RSI operation.

I_{QSBZN} A non-range RSI operation can be either long-running or non-long-running.

DRAFT

B5.2.2 Progress of long-running RSI operations

The information returned to the caller regarding the amount of progress which has been made differs between long-running RSI operations. This section describes the patterns which are used for reporting this progress.

B5.2.2.1 Long-running non-range RSI operation

I_{BWRWH} This pattern has the following characteristics:

- The operation is a non-range operation.
- The caller initiates the operation by calling an RSI “init” command. The input values to this command fully describe the requested operation.
- The caller continues the operation by calling an RSI “continue” command. The input values to this command include a handle which the implementation uses to retrieve a description of the operation.
- While the result of the “continue” command is RSI_INCOMPLETE, the caller calls the command again.
- While the operation is incomplete, no changes in state are observable by the caller.

The following pseudocode illustrate the programming model for this pattern.

```
int rsi_long_running_non_range(
    uint64_t in_value_1,
    uint64_t in_value_2,
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    ...
    uint64_t *out_value_1,
    uint64_t *out_value_2,
    ...)
{
    int result;

    result = RSI_OPERATION_INIT(
        in_value_1, in_value_2, ...,
        out_value_1, out_value_2, ...);
    if (result == RSI_SUCCESS) {
        do {
            result = RSI_OPERATION_CONTINUE(in_value_1);
        } while (result == RSI_INCOMPLETE);
    }

    return result;
}
```

See also:

- B5.4.2 *RSI_ATTESTATION_TOKEN_CONTINUE command*
- B5.4.3 *RSI_ATTESTATION_TOKEN_INIT command*

B5.2.2.2 Range RSI operation which returns progress address

I_{NDRGM}

This pattern has the following characteristics:

- The operation modifies the state of objects which are contiguous within an address space.
- The caller initiates the operation by calling an RSI command, with the target address range identified by input values `base` and `top`.
- If the result of this command is RSI_SUCCESS then an `out_top` output value indicates the top of the target address range for which the requested state change has been completed.
- In order to continue an incomplete operation, the call invokes the same command repeatedly, each time adjusting using the previous `out_top` as the new `base` value. This process is repeated until `out_top == top`.
- While the operation is incomplete, changes in state are observable by the caller and are explicitly reported in the `out_top` value.

The following pseudocode illustrate the programming model for this pattern.

```
int rsi_range_progress(
    uint64_t base, // input value 1
    uint64_t top,  // input value 2
    uint64_t in_value_3,
    ...
    uint64_t *out_top, // output value 1
    uint64_t *out_value_2,
    ...)
{
    int result;

    do {
        result = RSI_DO_OPERATION(
            base, top, in_value_3, ...,
            out_top, out_value_2, ...);
        base = *out_top;

        // If result == RSI_SUCCESS then the requested state change has been
        // applied to the range [base, *out_top).
```

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
    } while (result == RSI_SUCCESS && *out_top != top)

    // If result == RSI_SUCCESS && *out_top == top then the requested state change
    // has been applied to the entire range [base, top).

    return result;
}
```

I_{GSLYM} When a range RSI operation returns `out_top`, the requested state transition has been applied to all objects in the range `[base, out_top)`.

I_{XZZKG} When a range RSI operation returns `out_top`, the state of objects in the range `[out_top, top)` is unchanged by the operation.

See also:

- B5.4.7 *RSI_IPA_STATE_SET command*
- B5.4.11 *RSI_MEM_SET_PERM_INDEX command*
- B5.4.20 *RSI_VDEV_VALIDATE_MAPPING command*

B5.3 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:

- B5.5.3 *RsiCommandReturnCode type*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4 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_ARCH_DEV_ACTIVATE
0xC400019C	RSI_VDEV_DMA_ENABLE
0xC400019D	RSI_VDEV_GET_INFO
0xC400019F	RSI_VDEV_VALIDATE_MAPPING
0xC40001A0	RSI_MEM_GET_PERM_VALUE
0xC40001A1	RSI_MEM_SET_PERM_INDEX
0xC40001A2	RSI_MEM_SET_PERM_VALUE
0xC40001A3	RSI_PLANE_ENTER
0xC40001A4	RSI_VDEV_DMA_DISABLE
0xC40001AE	RSI_PLANE_SYSREG_READ
0xC40001AF	RSI_PLANE_SYSREG_WRITE

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.1 RSI_ARCH_DEV_ACTIVATE command

Activate an architectural device.

See also:

- A9.8.4 *VSMMU validation*

B5.4.1.1 Interface

B5.4.1.1.1 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
dev_type	X2	63:0	RsiArchDevType	Device type

B5.4.1.1.2 Context

The RSI_ARCH_DEV_ACTIVATE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm

walk	RmmRttWalkResult	`RttWalk(`
		`realm, base,`
		`RMM_RTT_PAGE_LEVEL,`
		`RMM_RTT_TREE_PRIMARY)`
			false	RTT walk result

vsmmu	RmmVsmmu	`VsmmuAt(walk.rtte.addr)`	false	VSMMU

B5.4.1.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

B5.4.1.2 Failure conditions

ID	Condition
base_align	`pre: !AddrIsRsiGranuleAligned(base)` `post: result == RSI_ERROR_INPUT`
base_bound	`pre: !AddrIsProtected(base, realm)` `post: result == RSI_ERROR_INPUT`
rtte_state	`pre: walk.rtte.state != RTTE_ARCH_DEV` `post: result == RSI_ERROR_INPUT`
vsmmu_state	`pre: vsmmu.state == VSMMU_ACTIVE` `post: result == RSI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.1.2.1 Failure condition ordering

The RSI_ARCH_DEV_ACTIVATE command does not have any failure condition orderings.

B5.4.1.3 Success conditions

ID	Condition
ripas	`post: RIPAS of entire address range of the device is equal to RIPAS_DEV.`
state	`post: vsmmu.state == VSMMU_ACTIVE`

B5.4.1.4 Footprint

The RSI_ARCH_DEV_ACTIVATE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.2 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*
- B5.2.2.1 *Long-running non-range RSI operation*
- B5.4.3 *RSI_ATTESTATION_TOKEN_INIT command*

B5.4.2.1 Interface

B5.4.2.1.1 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

B5.4.2.1.2 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

B5.4.2.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
len	X1	63:0	UInt64	Number of bytes written to buffer

B5.4.2.2 Failure conditions

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
addr_align	pre: !AddrIsRsiGranuleAligned(addr) post: result == RSI_ERROR_INPUT
addr_bound	pre: !AddrIsProtected(addr, realm) post: result == RSI_ERROR_INPUT
addr_empty	pre: walk.rtte.ripas == RIPAS_EMPTY post: result == RSI_ERROR_INPUT
offset_bound	pre: offset >= RSI_GRANULE_SIZE post: result == RSI_ERROR_INPUT
size_overflow	pre: offset + size < offset post: result == RSI_ERROR_INPUT
size_bound	pre: offset + size > RSI_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

B5.4.2.2.1 Failure condition ordering

The RSI_ATTESTATION_TOKEN_CONTINUE command does not have any failure condition orderings.

B5.4.2.3 Success conditions

ID	Condition
len	post: len == AttestationTokenWrite(addr, offset, size)
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

B5.4.2.4 Footprint

ID	Value
state	rec.attest_state

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.3 RSI_ATTESTATION_TOKEN_INIT command

Initialize the operation to retrieve an attestation token.

See also:

- A7.2 *Realm attestation*
- B5.2.2.1 *Long-running non-range RSI operation*
- B5.4.2 *RSI_ATTESTATION_TOKEN_CONTINUE command*

B5.4.3.1 Interface

B5.4.3.1.1 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

DRAFT

B5.4.3.1.2 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

B5.4.3.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
size	X1	63:0	UInt64	Upper bound on attestation token size in bytes

B5.4.3.2 Failure conditions

The RSI_ATTESTATION_TOKEN_INIT command does not have any failure conditions.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.3.3 Success conditions

ID	Condition
state	`post: rec.attest_state == ATTEST_IN_PROGRESS`
challenge	`post: rec.attest_challenge == ( ((challenge_0 :: challenge_1) :: (challenge_2 :: challenge_3)) :: ((challenge_4 :: challenge_5) :: (challenge_6 :: challenge_7)) )`
size	`post: size == AttestationTokenMaxSize(realm)`

B5.4.3.4 Footprint

ID	Value
state	`rec.attest_state`
challenge	`rec.attest_challenge`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.4 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*

B5.4.4.1 Interface

B5.4.4.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000191`
index	X1	63:0	UInt64	Feature register index

B5.4.4.1.2 Context

The RSI_FEATURES command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm

DRAFT

B5.4.4.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
value	X1	63:0	Bits64	Feature register value

B5.4.4.2 Failure conditions

The RSI_FEATURES command does not have any failure conditions.

B5.4.4.3 Success conditions

ID	Condition
value	`post: value == RsiFeatureRegisterEncode(realm, index)`

B5.4.4.4 Footprint

The RSI_FEATURES command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.5 RSI_HOST_CALL command

Make a Host call.

See also:

- A4.5 *Host call*
- A5.2.4 *RSI command access to a Protected IPA*

B5.4.5.1 Interface

B5.4.5.1.1 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

B5.4.5.1.2 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

DRAFT

B5.4.5.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

B5.4.5.2 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 == RIPAS_EMPTY` `post: result == RSI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.5.2.1 Failure condition ordering

The RSI_HOST_CALL command does not have any failure condition orderings.

B5.4.5.3 Success conditions

The RSI_HOST_CALL command does not have any success conditions.

B5.4.5.4 Footprint

ID	Value
gprs	`data.gprs`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.6 RSI_IPA_STATE_GET command

Get RIPAS of a target IPA range.

See also:

- A5.2 *Realm view of memory management*
- B5.4.7 *RSI_IPA_STATE_SET command*

B5.4.6.1 Interface

B5.4.6.1.1 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

B5.4.6.1.2 Context

The RSI_IPA_STATE_GET command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm

DRAFT

B5.4.6.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
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 RIPAS_DESTROYED without the Realm taking any action.

See also:

- A5.2.6 *Changes to RIPAS while Realm state is REALM_ACTIVE*

B5.4.6.2 Failure conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
base_align	pre: !AddrIsRsiGranuleAligned(base) post: result == RSI_ERROR_INPUT
end_align	pre: !AddrIsRsiGranuleAligned(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

B5.4.6.2.1 Failure condition ordering

The RSI_IPA_STATE_GET command does not have any failure condition orderings.

B5.4.6.3 Success conditions

ID	Condition
ripas	post: Value of out_top is such that RIPAS of address range [base, out_top) is equal to ripas.

B5.4.6.4 Footprint

The RSI_IPA_STATE_GET command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.7 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*
- B5.2.2.2 *Range RSI operation which returns progress address*
- B5.4.6 *RSI_IPA_STATE_GET command*

B5.4.7.1 Interface

B5.4.7.1.1 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].

If ripas is not RIPAS_RAM then flags.destroyed is ignored.

B5.4.7.1.2 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

B5.4.7.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
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_RESPONSE_REJECT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.7.2 Failure conditions

ID	Condition
base_align	`pre: !AddrIsRsiGranuleAligned(base)` `post: result == RSI_ERROR_INPUT`
top_align	`pre: !AddrIsRsiGranuleAligned(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_RIPAS_EMPTY) && (ripas != RSI_RIPAS_RAM)` `post: result == RSI_ERROR_INPUT`

B5.4.7.2.1 Failure condition ordering

The RSI_IPA_STATE_SET command does not have any failure condition orderings.

B5.4.7.3 Success conditions

ID	Condition
ripas	`post: RIPAS of address range [base, new_base) is equal to ripas.`
new_base	`post: new_base == rec.ripas_addr`
response	`post: response == RecRipasResponseToRsi(rec)`

B5.4.7.4 Footprint

The RSI_IPA_STATE_SET command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.8 RSI_MEASUREMENT_EXTEND command

Extend Realm Extensible Measurement (REM) value.

B5.4.8.1 Interface

B5.4.8.1.1 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

B5.4.8.1.2 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.rem[[index]]`	true	Previous measurement value

B5.4.8.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

B5.4.8.2 Failure conditions

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
index_bound	`pre: index < 1 || index > 4` `post: result == RSI_ERROR_INPUT`
size_bound	`pre: size > 64` `post: result == RSI_ERROR_INPUT`

B5.4.8.2.1 Failure condition ordering

The RSI_MEASUREMENT_EXTEND command does not have any failure condition orderings.

B5.4.8.3 Success conditions

ID	Condition

realm_meas	`post: realm.rem[[index - 1]] == 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)`

B5.4.8.4 Footprint

ID	Value
realm_meas	`realm.rem[[index - 1]]`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.9 RSI_MEASUREMENT_READ command

Read measurement for the current Realm.

See also:

- A7.1 *Realm measurements*
- D1.2.1 *Realm creation flow*

B5.4.9.1 Interface

B5.4.9.1.1 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.

B5.4.9.1.2 Context

The RSI_MEASUREMENT_READ command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm
meas	RmmRealmMeasurement	`RsiRealmMeasurement(realm, index)`	false	Measurement

DRAFT

B5.4.9.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
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”

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Register	Bits	Type	Description
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.

B5.4.9.2 Failure conditions

ID	Condition
index_bound	`pre: index > 4` `post: result == RSI_ERROR_INPUT`

B5.4.9.3 Success conditions

ID	Condition

sha_256	`pre: realm.hash_algo == HASH_SHA_256`
	`post: (value_0 == RealmMeasurementEncode(meas)[[0]]`
	`&& value_1 == RealmMeasurementEncode(meas)[[1]]`
	`&& value_2 == RealmMeasurementEncode(meas)[[2]]`
	`&& value_3 == RealmMeasurementEncode(meas)[[3]]`
	`&& value_4 == Zeros{64}()`
	`&& value_5 == Zeros{64}()`
	`&& value_6 == Zeros{64}()`
	`&& value_7 == Zeros{64}())`

sha_512	`pre: realm.hash_algo == HASH_SHA_512`
	`post: (value_0 == RealmMeasurementEncode(meas)[[0]]`
	`&& value_1 == RealmMeasurementEncode(meas)[[1]]`
	`&& value_2 == RealmMeasurementEncode(meas)[[2]]`
	`&& value_3 == RealmMeasurementEncode(meas)[[3]]`
	`&& value_4 == RealmMeasurementEncode(meas)[[4]]`
	`&& value_5 == RealmMeasurementEncode(meas)[[5]]`
	`&& value_6 == RealmMeasurementEncode(meas)[[6]]`
	`&& value_7 == RealmMeasurementEncode(meas)[[7]])`

B5.4.9.4 Footprint

The RSI_MEASUREMENT_READ command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.10 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*

B5.4.10.1 Interface

B5.4.10.1.1 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

B5.4.10.1.2 Context

The RSI_MEM_GET_PERM_VALUE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm

B5.4.10.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
value	X1	63:0	Bits64	Memory permission value

B5.4.10.2 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`

B5.4.10.2.1 Failure condition ordering

The RSI_MEM_GET_PERM_VALUE command does not have any failure condition orderings.

B5.4.10.3 Success conditions

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
label	`post: value == realm.overlay_perms[[plane_index]].values[[perm_index]]`

B5.4.10.4 Footprint

The RSI_MEM_GET_PERM_VALUE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.11 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*
- B5.2.2.2 *Range RSI operation which returns progress address*

B5.4.11.1 Interface

B5.4.11.1.1 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
handle	X4	63:0	Bits64	Handle value

B5.4.11.1.2 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

DRAFT

B5.4.11.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
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_handle	X3	63:0	Bits64	New handle value

The following unused bits of RSI_MEM_SET_PERM_INDEX output values MBZ: X2[63:1].

B5.4.11.2 Failure conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
base_align	pre: !AddrIsRsiGranuleAligned(base) post: result == RSI_ERROR_INPUT
top_align	pre: !AddrIsRsiGranuleAligned(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
handle	pre: Handle is invalid post: result == RSI_ERROR_INPUT

B5.4.11.2.1 Failure condition ordering

The RSI_MEM_SET_PERM_INDEX command does not have any failure condition orderings.

B5.4.11.3 Success conditions

ID	Condition
locked	post: realm.overlay_locked[[perm_index]] == MEM_PERM_LOCKED
new_base	post: new_base == rec.s2ap_addr
response	post: response == RecS2APResponseToRsi(rec)
new_handle	post: New handle is generated

B5.4.11.4 Footprint

ID	Value
locked	realm.overlay_locked[[perm_index]]

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.12 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*

B5.4.12.1 Interface

B5.4.12.1.1 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

B5.4.12.1.2 Context

The RSI_MEM_SET_PERM_VALUE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm

B5.4.12.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

B5.4.12.2 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`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.12.2.1 Failure condition ordering

The RSI_MEM_SET_PERM_VALUE command does not have any failure condition orderings.

B5.4.12.3 Success conditions

ID	Condition
label	`post: realm.overlay_perms[[plane_index]].values[[perm_index]] == value`

B5.4.12.4 Footprint

ID	Value
locked	`realm.overlay_perms[[plane_index]].values[[perm_index]]`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.13 RSI_PLANE_ENTER command

Enter a Plane.

See also:

- A5.2.4 *RSI command access to a Protected IPA*
- A10.2 *Planes exception model*

B5.4.13.1 Interface

B5.4.13.1.1 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

B5.4.13.1.2 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

DRAFT

B5.4.13.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

B5.4.13.2 Failure conditions

ID	Condition

idx_bound	`pre: (plane_idx == 0`
	`  || plane_idx > realm.num_aux_planes)`
	`post: result == RSI_ERROR_INPUT`

run_align	`pre: !AddrIsRsiGranuleAligned(run_ptr)` `post: result == RSI_ERROR_INPUT`
run_bound	`pre: !AddrIsProtected(run_ptr, realm)` `post: result == RSI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
run_empty	pre: walk.rtte.ripas == RIPAS_EMPTY post: result == RSI_ERROR_INPUT
el	pre: run.enter.pstate[3] == '1' post: result == RSI_ERROR_INPUT

B5.4.13.2.1 Failure condition ordering

The RSI_PLANE_ENTER command does not have any failure condition orderings.

B5.4.13.3 Success conditions

ID	Condition
plane_exit	post: run.exit contains Plane exit syndrome information.

B5.4.13.4 Footprint

The RSI_PLANE_ENTER command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.14 RSI_PLANE_SYSREG_READ command

Read a Plane register.

See also:

- A10.2.6 *Pn system registers*

B5.4.14.1 Interface

B5.4.14.1.1 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
addr	X2	63:0	RsiSysregAddress	System register address

The encoding value is an architecturally-defined system register encoding.

See also:

- *Arm Architecture Reference Manual for A-Profile architecture* [3]

B5.4.14.1.2 Context

The RSI_PLANE_SYSREG_READ command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm
rec	RmmRec	`CurrentRec()`	false	Current REC

DRAFT

B5.4.14.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
value_low	X1	63:0	Bits64	Lower 64 bits of system register value
value_high	X2	63:0	Bits64	Upper 64 bits of system register value

B5.4.14.2 Failure conditions

ID	Condition
idx_bound	`pre: plane_idx > realm.num_aux_planes` `post: result == RSI_ERROR_INPUT`
sysreg_valid	`pre: !PlaneSysregValid(rec, addr, RMM_READ)` `post: result == RSI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.14.2.1 Failure condition ordering

The RSI_PLANE_SYSREG_READ command does not have any failure condition orderings.

B5.4.14.3 Success conditions

ID	Condition

value_64	`pre: addr.d128 == RSI_FALSE`
	`post: (Zeros{64}() :: value_low) ==`
	`PlaneSysregValue(rec, plane_idx, addr)`

value_128	`pre: addr.d128 == RSI_TRUE`
	`post: (value_high :: value_low) ==`
	`PlaneSysregValue(rec, plane_idx, addr)`

B5.4.14.4 Footprint

The RSI_PLANE_SYSREG_READ command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.15 RSI_PLANE_SYSREG_WRITE command

Write a Plane register.

See also:

- A10.2.6 *Pn system registers*

B5.4.15.1 Interface

B5.4.15.1.1 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
addr	X2	63:0	RsiSysregAddress	System register address
value_low	X3	63:0	Bits64	Lower 64 bits of system register value
value_high	X4	63:0	Bits64	Upper 64 bits of system register value

The encoding value is an architecturally-defined system register encoding.

See also:

- *Arm Architecture Reference Manual for A-Profile architecture* [3]

B5.4.15.1.2 Context

The RSI_PLANE_SYSREG_WRITE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm
rec	RmmRec	`CurrentRec()`	false	Current REC

B5.4.15.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

B5.4.15.2 Failure conditions

ID	Condition
idx_bound	`pre: plane_idx > realm.num_aux_planes` `post: result == RSI_ERROR_INPUT`
sysreg_valid	`pre: !PlaneSysregValid(rec, addr, RMM_WRITE)` `post: result == RSI_ERROR_INPUT`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.15.2.1 Failure condition ordering

The RSI_PLANE_SYSREG_WRITE command does not have any failure condition orderings.

B5.4.15.3 Success conditions

ID	Condition
value_low	`post: PlaneSysregValue(rec, plane_idx, addr)[63:0] == value_low`
value_high	`pre: addr.d128 == RSI_TRUE` `post: PlaneSysregValue(rec, plane_idx, addr)[127:64] == value_high`

B5.4.15.4 Footprint

ID	Value
rec_sysregs	`rec.sysregs`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.16 RSI_REALM_CONFIG command

Read configuration for the current Realm.

See also:

- A5.2.4 *RSI command access to a Protected IPA*

B5.4.16.1 Interface

B5.4.16.1.1 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

B5.4.16.1.2 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

DRAFT

B5.4.16.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

B5.4.16.2 Failure conditions

ID	Condition
addr_align	`pre: !AddrIsRsiGranuleAligned(addr)` `post: result == RSI_ERROR_INPUT`
addr_bound	`pre: !AddrIsProtected(addr, realm)` `post: result == RSI_ERROR_INPUT`
addr_empty	`pre: walk.rtte.ripas == RIPAS_EMPTY` `post: result == RSI_ERROR_INPUT`

B5.4.16.2.1 Failure condition ordering

The RSI_REALM_CONFIG command does not have any failure condition orderings.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.16.3 Success conditions

ID	Condition
ipa_width	`post: cfg.ipa_width == realm.ipa_width`
hash_algo	`post: Equal(cfg.hash_algo, realm.hash_algo)`
num_aux_planes	`post: cfg.num_aux_planes == realm.num_aux_planes`
ats_plane	`post: cfg.ats_plane == realm.ats_plane`

B5.4.16.4 Footprint

The RSI_REALM_CONFIG command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.17 RSI_VDEV_DMA_DISABLE command

Disable DMA.

B5.4.17.1 Interface

B5.4.17.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC40001A4`
vdev_id	X1	63:0	Bits64	Realm device identifier

B5.4.17.1.2 Context

The RSI_VDEV_DMA_DISABLE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm

vdev	RmmVdev	`VdevFromVdevId(`
		`  realm,`
		`  vdev_id)`
			false	Realm device

B5.4.17.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

DRAFT

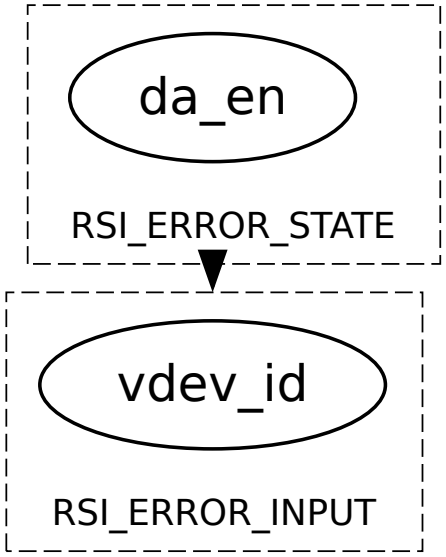
B5.4.17.2 Failure conditions

ID	Condition
da_en	`pre:  realm.feat_da != FEATURE_TRUE` `post: result == RSI_ERROR_STATE`
vdev_id	`pre:  VdevIdIsFree(realm, vdev_id)` `post: result == RSI_ERROR_INPUT`

B5.4.17.2.1 Failure condition ordering

```
[da_en] < [vdev_id]
```


Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



B5.4.17.3 Success conditions

ID	Condition
dma_state	`post: vdev.dma_state == VDEV_DMA_DISABLED`

B5.4.17.4 Footprint

The RSI_VDEV_DMA_DISABLE command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.18 RSI_VDEV_DMA_ENABLE command

Enable DMA.

B5.4.18.1 Interface

B5.4.18.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400019C`
vdev_id	X1	63:0	Bits64	Realm device identifier
flags	X2	63:0	RsiVdevDmaFlags	Flags
non_ats_plane	X3	63:0	UInt64	Index of Plane whose stage 2 permissions are observed by non-ATS requests from the device
lock_nonce	X4	63:0	UInt64	Nonce generated on most recent transition to LOCKED state
meas_nonce	X5	63:0	UInt64	GET_MEASUREMENT request sequence number
report_nonce	X6	63:0	UInt64	GET_INTERFACE_REPORT request sequence number

B5.4.18.1.2 Context

The RSI_VDEV_DMA_ENABLE command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm

vdev	RmmVdev	`VdevFromVdevId(`
		`  realm,`
		`  vdev_id)`
			false	Realm device

B5.4.18.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

B5.4.18.2 Failure conditions

ID	Condition
da_en	`pre:  realm.feat_da != FEATURE_TRUE` `post: result == RSI_ERROR_STATE`
vdev_id	`pre:  VdevIdIsFree(realm, vdev_id)` `post: result == RSI_ERROR_INPUT`

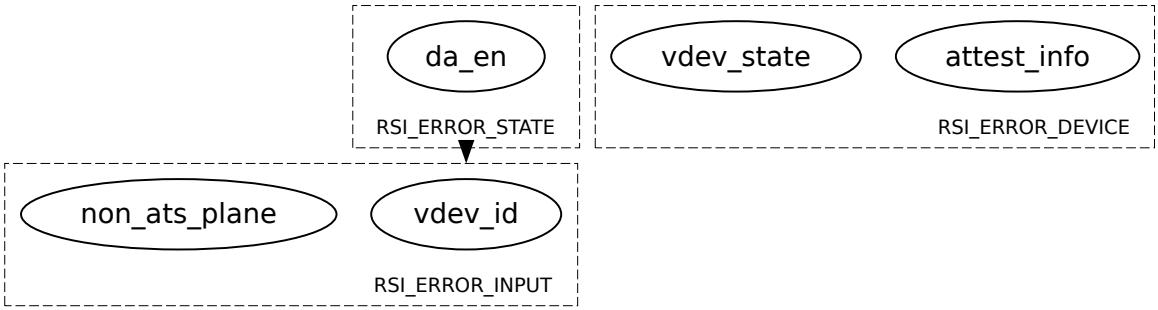
DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
non_ats_plane	pre: (non_ats_plane == 0 || non_ats_plane > realm.num_aux_planes) post: result == RSI_ERROR_INPUT
attest_info	pre: !VdevAttestInfoEqual(lock_nonce, meas_nonce, report_nonce, vdev.attest_info) post: result == RSI_ERROR_DEVICE
vdev_state	pre: vdev.vdev_state != VDEV_STARTED post: result == RSI_ERROR_DEVICE

B5.4.18.2.1 Failure condition ordering

```
[da_en] < [vdev_id, non_ats_plane]
```



B5.4.18.3 Success conditions

ID	Condition
dma_state	post: vdev.dma_state == VDEV_DMA_ENABLED
non_ats_plane	post: vdev.non_ats_plane == non_ats_plane

B5.4.18.4 Footprint

The RSI_VDEV_DMA_ENABLE command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.19 RSI_VDEV_GET_INFO command

Get information for a device.

Device configuration information, including digests of attestation evidence for the device are written to an RsiVdevInfo 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*
- A9.6.1 *Realm retrieval of device attestation evidence*
- B5.4.16 *RSI_REALM_CONFIG command*
- B5.5.28 *RsiVdevInfo type*

B5.4.19.1 Interface

B5.4.19.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400019D`
vdev_id	X1	63:0	Bits64	Realm device identifier
addr	X2	63:0	Address	IPA to which the configuration data will be written

B5.4.19.1.2 Context

The RSI_VDEV_GET_INFO command operates on the following context.

Name	Type	Value	Before	Description
realm	RmmRealm	`CurrentRealm()`	false	Current Realm

vdev	RmmVdev	`VdevFromVdevId(`
		`  realm,`
		`  vdev_id)`
			false	Realm device

pdev	RmmPdev	`PdevAt(vdev.pdev)`	false	Physical device
cfg	RsiVdevInfo	`RsiVdevInfoAt(addr)`	false	Device configuration

walk	RmmRttWalkResult	`RttWalk(`
		`  realm, addr,`
		`  RMM_RTT_PAGE_LEVEL,`
		`  RMM_RTT_TREE_PRIMARY)`
			false	RTT walk result

B5.4.19.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result

DRAFT

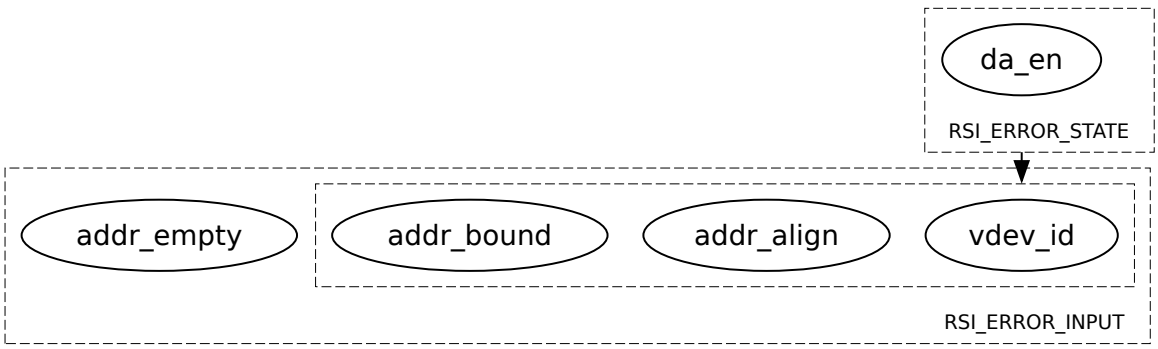
Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.19.2 Failure conditions

ID	Condition
da_en	pre: realm.feat_da != FEATURE_TRUE post: result == RSI_ERROR_STATE
vdev_id	pre: VdevIdIsFree(realm, vdev_id) post: result == RSI_ERROR_INPUT
addr_align	pre: !AddrIsAligned(addr, 512) post: result == RSI_ERROR_INPUT
addr_bound	pre: !AddrIsProtected(addr, realm) post: result == RSI_ERROR_INPUT
addr_empty	pre: walk.rtte.ripas == RIPAS_EMPTY post: result == RSI_ERROR_INPUT

B5.4.19.2.1 Failure condition ordering

```
[da_en] < [vdev_id, addr_align, addr_bound]
```



B5.4.19.3 Success conditions

ID	Condition
hash_algo	post: Equal(cfg.hash_algo, pdev.hash_algo)
attest_info	post: VdevAttestInfoEqual(cfg.lock_nonce, cfg.meas_nonce, cfg.report_nonce, vdev.attest_info)
protocol_data_digest	post: cfg.protocol_data_digest == pdev.protocol_data_digest
meas_digest	post: cfg.meas_digest == vdev.meas_digest
report_digest	post: cfg.report_digest == vdev.report_digest
state	post: cfg.state == VdevStateToRsi(vdev.vdev_state)

B5.4.19.4 Footprint

The RSI_VDEV_GET_INFO command does not have any footprint.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.20 RSI_VDEV_VALIDATE_MAPPING command

Validate Realm device memory mappings.

See also:

- A9.6.2 *Realm validation of device memory mappings*
- B5.2.2.2 *Range RSI operation which returns progress address*

B5.4.20.1 Interface

B5.4.20.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC400019F`
vdev_id	X1	63:0	Bits64	Realm device identifier
ipa_base	X2	63:0	Address	Base of target IPA region
ipa_top	X3	63:0	Address	Top of target IPA region
pa_base	X4	63:0	Address	Base of target PA region
flags	X5	63:0	RsiDevMemFlags	Flags
lock_nonce	X6	63:0	UInt64	Nonce generated on most recent transition to LOCKED state
meas_nonce	X7	63:0	UInt64	GET_MEASUREMENT request sequence number
report_nonce	X8	63:0	UInt64	GET_INTERFACE_REPORT request sequence number

DRAFT

B5.4.20.1.2 Context

The RSI_VDEV_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

vdev	RmmVdev	`VdevFromVdevId(`
		`  realm,`
		`  vdev_id)`
			false	Realm device

B5.4.20.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
new_ipa_base	X1	63:0	Address	Base of IPA region which was not modified by the command

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Register	Bits	Type	Description
response	X2	0:0	RsiResponse	Whether the Host accepted or rejected the request

The following unused bits of RSI_VDEV_VALIDATE_MAPPING output values MBZ: X2[63:1].

B5.4.20.2 Failure conditions

ID	Condition
da_en	`pre: realm.feat_da != FEATURE_TRUE` `post: result == RSI_ERROR_STATE`
vdev_id	`pre: VdevIdIsFree(realm, vdev_id)` `post: result == RSI_ERROR_INPUT`

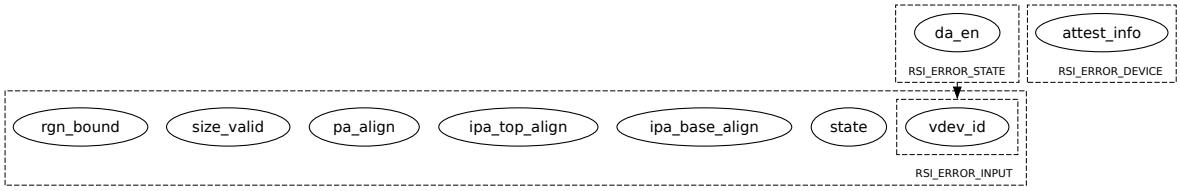
state	`pre: (vdev.vdev_state != VDEV_LOCKED`
	`&& vdev.vdev_state != VDEV_STARTED)`
	`post: result == RSI_ERROR_INPUT`

ipa_base_align	`pre: !AddrIsRsiGranuleAligned(ipa_base)` `post: result == RSI_ERROR_INPUT`
ipa_top_align	`pre: !AddrIsRsiGranuleAligned(ipa_top)` `post: result == RSI_ERROR_INPUT`
pa_align	`pre: !AddrIsRsiGranuleAligned(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`

attest_info	`pre: !VdevAttestInfoEqual(`
	`lock_nonce,`
	`meas_nonce,`
	`report_nonce,`
	`vdev.attest_info)`
	`post: result == RSI_ERROR_DEVICE`

B5.4.20.2.1 Failure condition ordering

```
[da_en] < [vdev_id]
```



B5.4.20.3 Success conditions

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
new_ipa_base	`post: new_ipa_base == rec.dev_mem_addr`
response	`post: response == RecDevMemResponseToRsi(rec)`

B5.4.20.4 Footprint

The RSI_VDEV_VALIDATE_MAPPING command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.21 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*

B5.4.21.1 Interface

B5.4.21.1.1 Input values

Name	Register	Bits	Type	Description
fid	X0	63:0	UInt64	FID, value `0xC4000190`
req	X1	63:0	RsiInterfaceVersion	Requested interface revision

B5.4.21.1.2 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
lower	X1	63:0	RsiInterfaceVersion	Lower supported interface revision
higher	X2	63:0	RsiInterfaceVersion	Higher supported interface revision

B5.4.21.2 Failure conditions

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

ID	Condition
incompat_lower	pre: (!RsiVersionIsSupported(req) && RsiVersionLowerIsSupported(req)) post: (result == RSI_ERROR_INPUT && VersionEqual(lower, RsiVersionHighestBelow(req)) && VersionEqual(higher, RsiVersionHighest()))
incompat_higher	pre: (!RsiVersionIsSupported(req) && !RsiVersionLowerIsSupported(req) && RsiVersionHigherIsSupported(req)) post: (result == RSI_ERROR_INPUT && VersionEqual(lower, higher) && VersionEqual(higher, RsiVersionHighest()))

B5.4.21.2.1 Failure condition ordering

The RSI_VERSION command does not have any failure condition orderings.

B5.4.21.3 Success conditions

ID	Condition
lower	post: VersionEqual(lower, req)
higher	post: VersionEqual(higher, RsiVersionHighest())

B5.4.21.4 Footprint

The RSI_VERSION command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.22 RSI_VSMMU_GET_INFO command

Get information of a VSMMU.

See also:

- A9.8.4 *VSMMU validation*

B5.4.22.1 Interface

B5.4.22.1.1 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

B5.4.22.1.2 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

DRAFT

B5.4.22.1.3 Output values

Name	Register	Bits	Type	Description
result	X0	63:0	RsiCommandReturnCode	Command result
top	X1	63:0	Address	Top IPA of the VSMMU

B5.4.22.2 Failure conditions

ID	Condition
addr_align	`pre:  !AddrIsRsiGranuleAligned(addr)` `post: result == RSI_ERROR_INPUT`
addr_bound	`pre:  !AddrIsProtected(addr, realm)` `post: result == RSI_ERROR_INPUT`
rtte_state	`pre:  walk.rtte.state != RTTE_ARCH_DEV` `post: result == RSI_ERROR_INPUT`
vsmmu_base	`pre:  addr != VsmmuAt(walk.rtte.addr).reg_base` `post: result == RSI_ERROR_INPUT`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.4.22.2.1 Failure condition ordering

The RSI_VSMMU_GET_INFO command does not have any failure condition orderings.

B5.4.22.3 Success conditions

ID	Condition
vsmmu_base	`post: top == VsmmuAt(walk.rtte.addr).reg_top`

B5.4.22.4 Footprint

The RSI_VSMMU_GET_INFO command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.5 RSI types

This section defines types which are used in the RSI interface.

B5.5.1 RsiArchDevType type

The RsiArchDevType enumeration represents architectural device type.

The RsiArchDevType enumeration is a concrete type.

The width of the RsiArchDevType enumeration is 64 bits.

The values of the RsiArchDevType enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_ARCH_DEV_SMMUV3	SMMUv3

Unused encodings for the RsiArchDevType enumeration are reserved for use by future versions of this specification.

B5.5.2 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.

DRAFT

Encoding	Name	Description
0	RSI_FALSE	False
1	RSI_TRUE	True

The RsiBoolean enumeration is used in the following types:

- RsiSysregAddress
- RsiVdevFlags

B5.5.3 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

B5.5.4 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.

DRAFT

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

B5.5.5 RsiDevMemFlags type

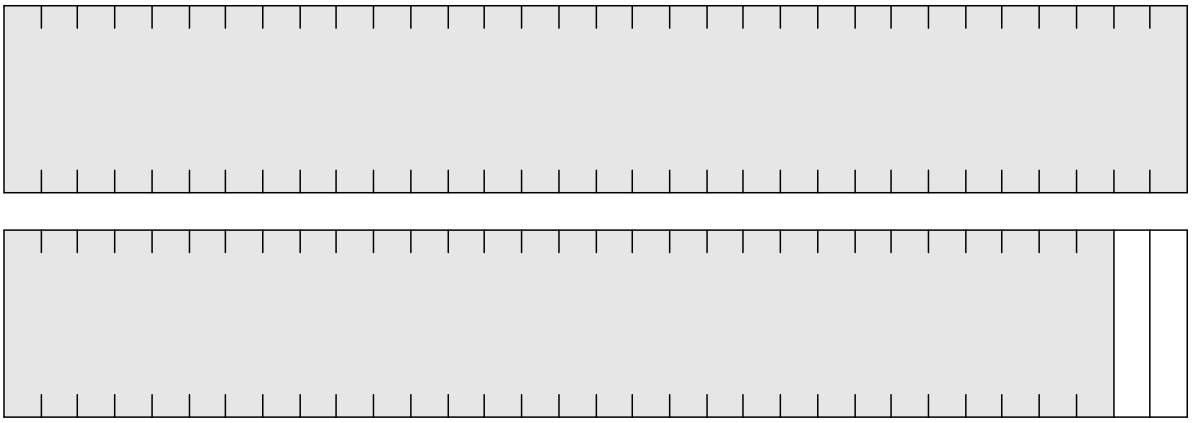
The RsiDevMemFlags fieldset contains flags which describe properties of a device memory mapping.

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



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

DRAFT

B5.5.6 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

B5.5.7 RsiFeature type

The RsiFeature enumeration represents whether a feature is enabled.

The RsiFeature enumeration is a concrete type.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- RsiVdevDmaFlags
- RsiFeatureRegister0
- RsiVdevFlags

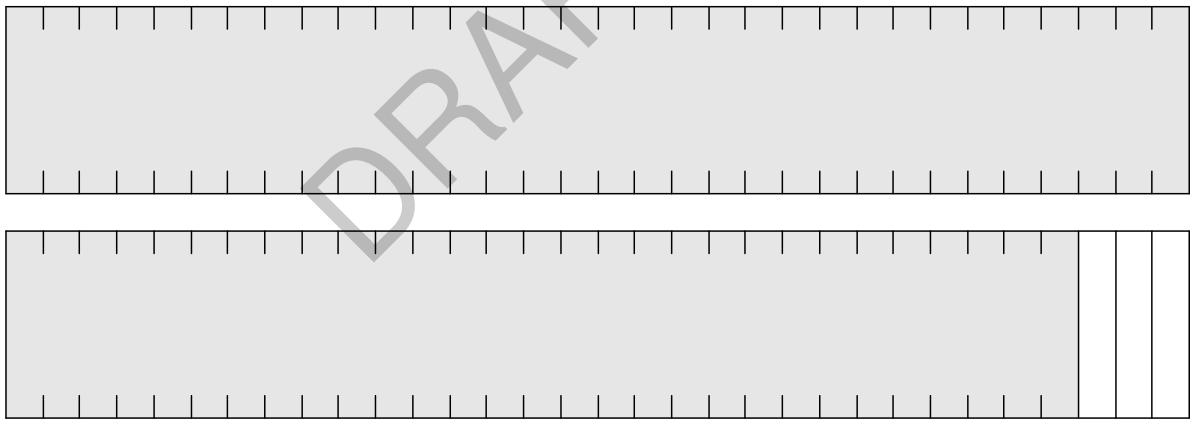
B5.5.8 RsiFeatureRegister0 type

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
ATS	2	Whether Address Translation Service is supported for devices assigned to the Realm	RsiFeature
	63:3	Reserved	MBZ

B5.5.9 RsiGicOwner type

The RsiGicOwner enumeration represents which Plane is GIC owner.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

B5.5.10 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:

- B5.4.16 *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)* [25])
1	RSI_HASH_SHA_512	SHA-512 (*Secure Hash Standard (SHS)* [25])
2	RSI_HASH_SHA_384	SHA-384 (*Secure Hash Standard (SHS)* [25])

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:

- RsiRealmConfig
- RsiVdevInfo

B5.5.11 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*
- B5.4.5 *RSI_HOST_CALL command*

The members of the RsiHostCall structure are shown in the following table.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
imm	0x0	UInt16	Immediate value
gprs[31]	0x8	Bits64	Registers

Unused bits of the RsiHostCall structure SBZ.

B5.5.12 RsiInterfaceVersion type

The RsiInterfaceVersion fieldset contains an RSI interface version.

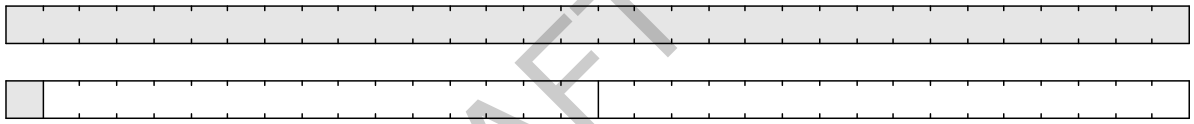
The RsiInterfaceVersion fieldset is a concrete type.

The width of the RsiInterfaceVersion fieldset is 64 bits.

See also:

- B5.1 *RSI version*
- B5.4.21 *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.

DRAFT

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

B5.5.13 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 (0x800) bytes.

The members of the RsiPlaneEnter structure are shown in the following table.

Name	Byte offset	Type	Description
flags	0x0	RsiPlaneEnterFlags	Flags
pc	0x8	Bits64	Program counter
pstate	0x10	Bits64	PSTATE

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
gprs[31]	0x100	Bits64	Registers
gicv3_hcr	0x200	Bits64	GICv3 Hypervisor Control Register value
gicv3_lrs[16]	0x208	Bits64	GICv3 List Register values
elr_el1	0x400	Bits64	ELR_EL1 value

Unused bits of the RsiPlaneEnter structure SBZ.

The RsiPlaneEnter structure is used in the following types:

- RsiPlaneRun

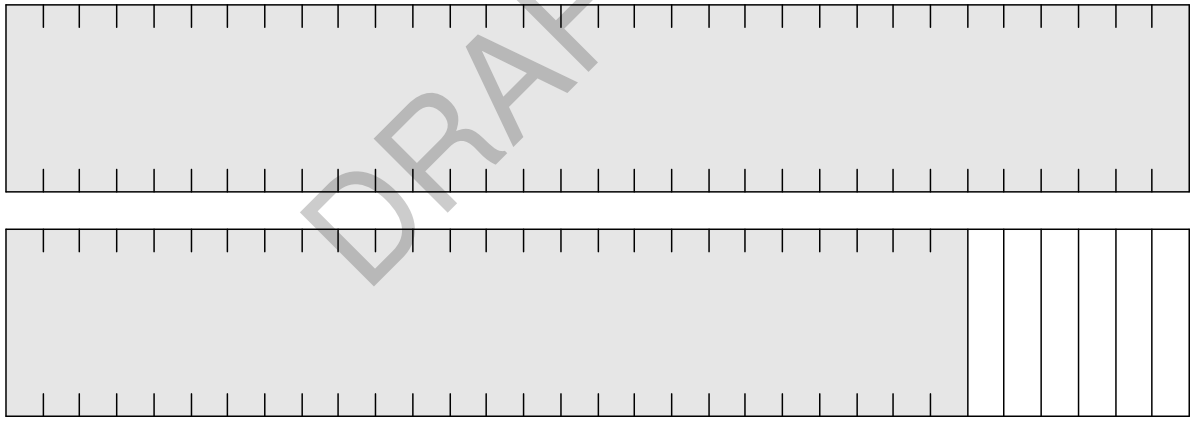
B5.5.14 RsiPlaneEnterFlags type

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.



DRAFT

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Bits	Description	Value
trap_simd	4	Whether to trap access to SIMD and SVE by the Plane.	RsiTrap
trap_dbg	5	Whether to trap debug exceptions taken from the Plane.	RsiTrap
	63:6	Reserved	SBZ

The RsiPlaneEnterFlags fieldset is used in the following types:

- RsiPlaneEnter

B5.5.15 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.

DRAFT

Name	Byte offset	Type	Description
reason	`0x0`	RsiPlaneExitReason	Exit reason
pc	`0x8`	Bits64	Program counter
pstate	`0x10`	Bits64	PSTATE
gprs[31]	`0x100`	Bits64	Registers
esr_el2	`0x200`	Bits64	Exception Syndrome Register
far_el2	`0x208`	Bits64	Fault Address Register
hpfar_el2	`0x210`	Bits64	Hypervisor IPA Fault Address register
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
sctlr_el1	`0x500`	Bits64	SCTLR_EL1 value

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
vbar_el1	0x508	Bits64	VBAR_EL1 value
elr_el1	0x510	Bits64	ELR_EL1 value
pmu_ovf_status	0x600	RsiPmuOverflowStatus	PMU overflow status

Unused bits of the RsiPlaneExit structure SBZ.

The RsiPlaneExit structure is used in the following types:

- RsiPlaneRun

B5.5.16 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
1	RSI_EXIT_IRQ	Plane exit due to IRQ
2	RSI_EXIT_HOST	Plane exit due to Host action

DRAFT

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

B5.5.17 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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B5.5.18 RsiPmuOverflowStatus type

The RsiPmuOverflowStatus enumeration represents PMU overflow status.

The RsiPmuOverflowStatus enumeration is a concrete type.

The width of the RsiPmuOverflowStatus enumeration is 8 bits.

The values of the RsiPmuOverflowStatus enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_PMU_OVERFLOW_NOT_ACTIVE	PMU overflow is not active.
1	RSI_PMU_OVERFLOW_ACTIVE	PMU overflow is active.

Unused encodings for the RsiPmuOverflowStatus enumeration are reserved for use by future versions of this specification.

The RsiPmuOverflowStatus enumeration is used in the following types:

- RsiPlaneExit

B5.5.19 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:

- B5.4.16 *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
ats_plane	`0x20`	UInt64	Index of Plane whose stage 2 permissions are observed by ATS requests from devices assigned to the Realm
rpv	`0x200`	Bits512	Realm Personalization Value

Unused bits of the RsiRealmConfig structure MBZ.

B5.5.20 RsiResponse type

The RsiResponse enumeration represents whether the Host accepted or rejected a Realm request.

The RsiResponse enumeration is a concrete type.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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_RESPONSE_ACCEPT	Host accepted the Realm request.
1	RSI_RESPONSE_REJECT	Host rejected the Realm request.

B5.5.21 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*
- B5.4.6 *RSI_IPA_STATE_GET command*
- B5.4.7 *RSI_IPA_STATE_SET command*

The values of the RsiRipas enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_RIPAS_EMPTY	Address where no Realm resources are mapped.
1	RSI_RIPAS_RAM	Address where private code or data owned by the Realm is mapped.
2	RSI_RIPAS_DESTROYED	Address which is inaccessible to the Realm due to an action taken by the Host.
3	RSI_RIPAS_DEV	Address where memory of an assigned Realm device is mapped.

DRAFT

Unused encodings for the RsiRipas enumeration are reserved for use by future versions of this specification.

B5.5.22 RsiRipasChangeDestroyed type

The RsiRipasChangeDestroyed enumeration represents whether a RIPAS change from RIPAS_DESTROYED to RIPAS_RAM 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 RIPAS_DESTROYED to RIPAS_RAM should not be permitted.
1	RSI_CHANGE_DESTROYED	A RIPAS change from RIPAS_DESTROYED to RIPAS_RAM should be permitted.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The RsiRipasChangeDestroyed enumeration is used in the following types:

- RsiRipasChangeFlags

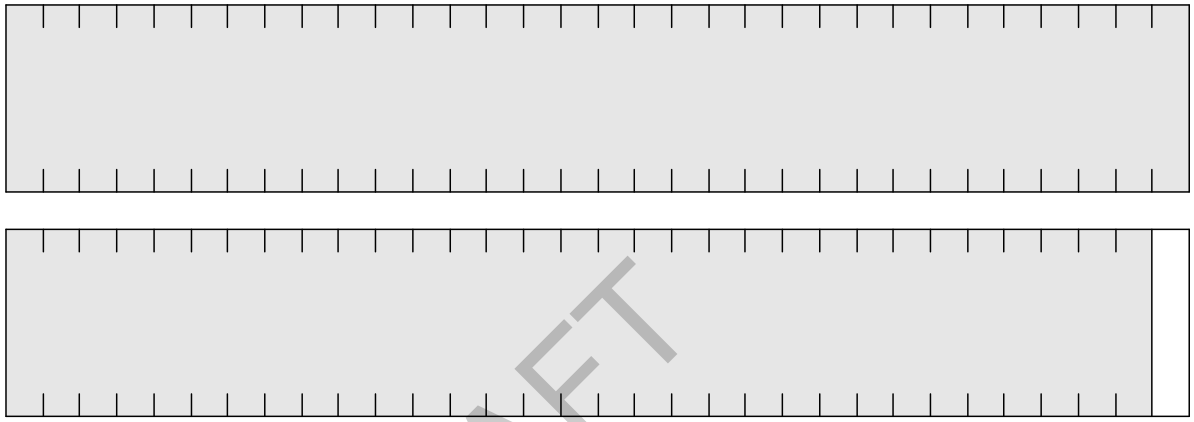
B5.5.23 RsiRipasChangeFlags type

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 RIPAS_DESTROYED to RIPAS_RAM should be permitted	RsiRipasChangeDestroyed
	63:1	Reserved	SBZ

B5.5.24 RsiSysregAddress type

The RsiSysregAddress fieldset contains system register address.

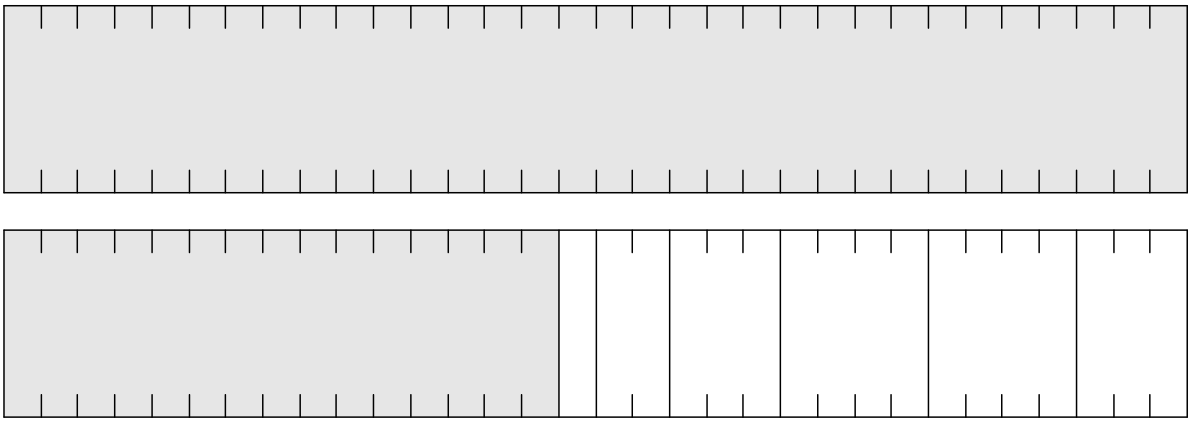
The RsiSysregAddress fieldset is a concrete type.

The width of the RsiSysregAddress fieldset is 64 bits.

The fields of the RsiSysregAddress fieldset are shown in the following diagram.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



The fields of the RsiSysregAddress fieldset are shown in the following table.

Name	Bits	Description	Value
Op2	2:0	Op2	Bits3
CRm	6:3	CRm	Bits4
CRn	10:7	CRn	Bits4
Op1	13:11	Op1	Bits3
Op0	15:14	Op0	Bits2
d128	16	Perform 128-bit sysreg access	RsiBoolean
	63:17	Reserved	SBZ

DRAFT

B5.5.25 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

B5.5.26 RsiVdevDmaFlags type

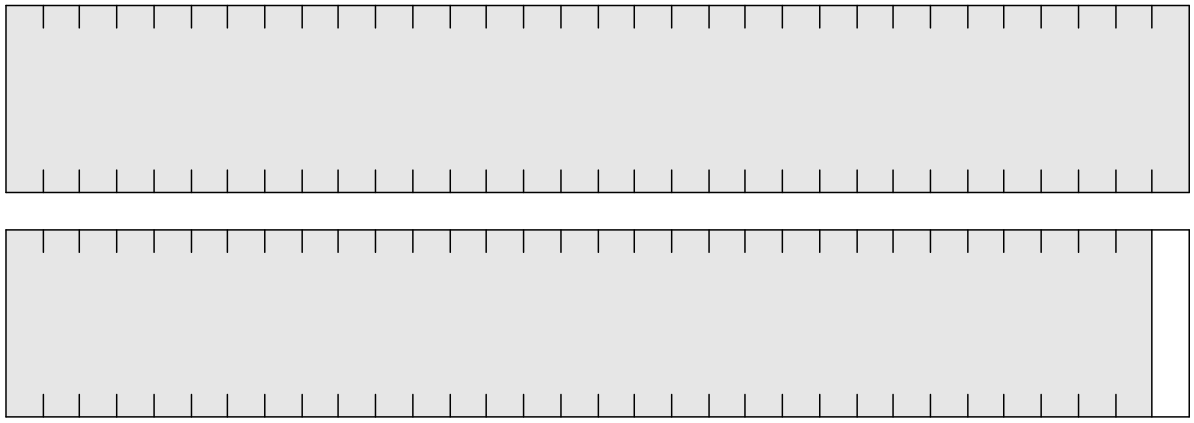
The RsiVdevDmaFlags fieldset contains flags which control device DMA.

The RsiVdevDmaFlags fieldset is a concrete type.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The width of the RsiVdevDmaFlags fieldset is 64 bits.

The fields of the RsiVdevDmaFlags fieldset are shown in the following diagram.



The fields of the RsiVdevDmaFlags fieldset are shown in the following table.

Name	Bits	Description	Value
ats	0	Whether to enable ATS for this device.	RsiFeature
	63:1	Reserved	SBZ

DRAFT

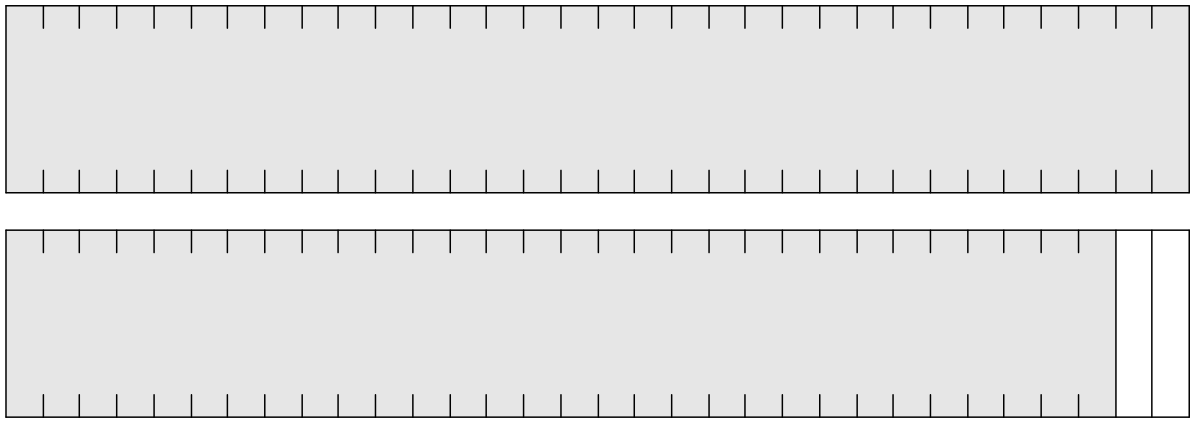
B5.5.27 RsiVdevFlags type

The RsiVdevFlags fieldset contains flags which describe properties of a device.

The RsiVdevFlags fieldset is a concrete type.

The width of the RsiVdevFlags fieldset is 64 bits.

The fields of the RsiVdevFlags fieldset are shown in the following diagram.



The fields of the RsiVdevFlags fieldset are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Bits	Description	Value
vsmmu	0	Whether this device is associated with a VSMMU	RsiFeature
protocol_data_set	1	Whether protocol data has been set for this device	RsiBoolean
	63:2	Reserved	MBZ

The RsiVdevFlags fieldset is used in the following types:

- RsiVdevInfo

B5.5.28 RsiVdevInfo type

The RsiVdevInfo structure contains device configuration information.

The RsiVdevInfo structure is a concrete type.

The width of the RsiVdevInfo structure is 512 (`0x200`) bytes.

See also:

- A9.6 *Realm management of an assigned virtual device*
- B5.4.19 *RSI_VDEV_GET_INFO command*

The members of the RsiVdevInfo structure are shown in the following table.

Name	Byte offset	Type	Description
flags	`0x0`	RsiVdevFlags	Flags
id_index	`0x8`	UInt64	Device identity index
hash_algo	`0x10`	RsiHashAlgorithm	Algorithm used to generate device digests
lock_nonce	`0x18`	UInt64	Nonce generated on most recent transition to LOCKED state
meas_nonce	`0x20`	UInt64	Nonce generated on most recent GET_MEASUREMENT request
report_nonce	`0x28`	UInt64	Nonce generated on most recent GET_INTERFACE_REPORT request
format_type	`0x30`	RsiVdevReportFormatType	Report format type
format_version	`0x38`	Bits64	Report format version If format_type is RSI_VDEV_REPORT_FORMAT_TDISP then this field specifies the TDISP version, encoded as follows: bits[31:16]: major version bits[15:0]: minor version
state	`0x40`	RsiVdevState	State of the device

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Byte offset	Type	Description
protocol_data_digest	0x80	Bits512	Protocol data digest If protocol_data_set is RSI_TRUE, this field contains the digest of the protocol data associated with this device; otherwise, this field is set to zero.
identity_digest	0xc0	Bits512	Identity digest
pubkey_digest	0x100	Bits512	Public key digest
meas_digest	0x140	Bits512	Measurement digest
report_digest	0x180	Bits512	Interface report digest
vsmmu_addr	0x1c0	Address	Base IPA of the VSMMU which is associated with this device. This field is valid only if flags.vsmmu == RSI_FEATURE_TRUE.
vsmmu_vsid	0x1c8	Bits64	Virtual Stream ID. This field is valid only if flags.vsmmu == RSI_FEATURE_TRUE.

Unused bits of the RsiVdevInfo structure MBZ.

B5.5.29 RsiVdevReportFormatType type

The RsiVdevReportFormatType enumeration represents device report format type.

The RsiVdevReportFormatType enumeration is a concrete type.

The width of the RsiVdevReportFormatType enumeration is 8 bits.

The values of the RsiVdevReportFormatType enumeration are shown in the following table.

Encoding	Name	Description
0	RSI_VDEV_REPORT_FORMAT_IMPDEF	The report format is IMPLEMENTATION DEFINED.
1	RSI_VDEV_REPORT_FORMAT_TDISP	The report format is TDISP.

Unused encodings for the RsiVdevReportFormatType enumeration are reserved for use by future versions of this specification.

The RsiVdevReportFormatType enumeration is used in the following types:

- RsiVdevInfo

B5.5.30 RsiVdevState type

The RsiVdevState enumeration represents the state of a VDEV.

The RsiVdevState enumeration is a concrete type.

The width of the RsiVdevState enumeration is 8 bits.

The values of the RsiVdevState enumeration are shown in the following table.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Encoding	Name	Description
0	RSI_VDEV_UNLOCKED	Device interface is unlocked.
1	RSI_VDEV_LOCKED	Device interface is locked.
2	RSI_VDEV_STARTED	Device interface is started.
3	RSI_VDEV_ERROR	Device interface has reported a fatal error.

Unused encodings for the RsiVdevState enumeration are reserved for use by future versions of this specification.

The RsiVdevState enumeration is used in the following types:

- RsiVdevInfo

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter B6
Power State Control Interface

DRAFT

This section describes how Power State Control Interface (PSCI) function execution by a Realm execution of SMC instructions is handled.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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)* [28]
- A2.4.2 *REC attributes*
- A4.3.7 *REC exit due to PSCI*
- A4.5 *Host call*
- D1.4 *PSCI flows*

B6.2 PSCI version

R_{TFCVF} The RMM must support version >= 1.1 of the Power State Control Interface.

See also:

- B6.3.8 *PSCI_VERSION command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B6.3 PSCI commands

The following table summarizes the FIDs of commands in the PSCI interface.

FID	Command
0xC4000000	PSCI_VERSION
0xC4000001	PSCI_CPU_SUSPEND
0xC4000002	PSCI_CPU_OFF
0xC4000003	PSCI_CPU_ON
0xC4000004	PSCI_AFFINITY_INFO
0xC4000008	PSCI_SYSTEM_OFF
0xC4000009	PSCI_SYSTEM_RESET
0xC400000A	PSCI_FEATURES

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- A2.4.2 *REC attributes*
- A4.3.7 *REC exit due to PSCI*
- B4.5.28 *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_leve l	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`

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B6.3.2 PSCI_CPU_OFF command

Power down the calling core.

This command causes a REC exit due to PSCI.

See also:

- A2.4.2 *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 `0xC4000002`

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- A2.4.2 *REC attributes*
- A4.3.7 *REC exit due to PSCI*
- B4.5.28 *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)

DRAFT

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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	post: target_rec.pc == ToBits64(UInt(entry_point_address))
runnable	post: target_rec.flags.runnable == RUNNABLE

B6.3.3.4 Footprint

ID	Value
runnable	target_rec.flags.runnable
pc	target_rec.pc

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 `0xC400000A`
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

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B6.3.6 PSCI_SYSTEM_OFF command

Shut down the system.

This command causes a REC exit due to PSCI.

See also:

- A2.4.2 *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 `0xC4000008`

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

DRAFT

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	`post: realm.state == REALM_SYSTEM_OFF`

Following execution of PSCI_SYSTEM_OFF, control does not return to the caller.

B6.3.6.4 Footprint

ID	Value
state	`realm.state`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B6.3.7 PSCI_SYSTEM_RESET command

Shut down the system.

This command causes a REC exit due to PSCI.

See also:

- A2.4.2 *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 `0xC4000009`

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

DRAFT

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	`post: realm.state == REALM_SYSTEM_OFF`

Following execution of PSCI_SYSTEM_RESET, control does not return to the caller.

B6.3.7.4 Footprint

ID	Value
state	`realm.state`

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 `0xC4000000`

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

ID	Condition
version	`post: VersionEqual(result, PsciVersion())`

B6.3.8.4 Footprint

The PSCI_VERSION command does not have any footprint.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

B6.4 PSCI types

This section defines types which are used in the PSCI interface.

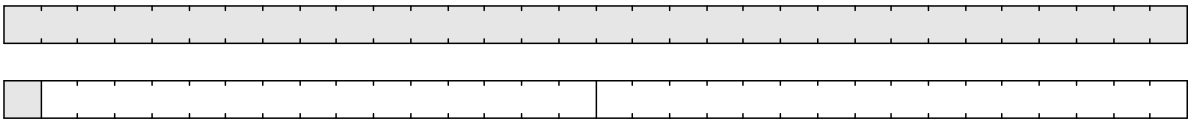
B6.4.1 PsciInterfaceVersion type

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

DRAFT

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

DRAFT

Part C
Constants and types

Chapter C1
RMM constants

DRAFT

This section describes constants which are used in the definition of RMM commands or RMM abstract state.

C1.1 RMM_NUM_PERM_OVERLAY_INDICES

Number of permission overlay indices.

The value of RMM_NUM_PERM_OVERLAY_INDICES is 15.

C1.2 RMM_RTT_PAGE_LEVEL

RTT level of a page entry.

The value of RMM_RTT_PAGE_LEVEL is 3.

C1.3 RMM_RTT_TREE_PRIMARY

Index of primary RTT tree.

The value of RMM_RTT_TREE_PRIMARY is 0.

C1.4 RSI_GRANULE_SIZE

Size of an RSI Granule in bytes.

The value of RSI_GRANULE_SIZE is 0x1000.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter C2
RMM types

DRAFT

This section describes types which are used to model the abstract state of the RMM.

C2.1 RmmAddrRange type

The RmmAddrRange structure contains address range.

The RmmAddrRange structure is an abstract type.

The members of the RmmAddrRange 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 RmmAddrRange structure is used in the following types:

- RmmPdevStream
- RmmVdev

C2.2 RmmBoolean type

The RmmBoolean enumeration represents whether a feature is enabled.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- RmmPdevStreamResult
- RmmRttS2APDirect
- RmmVdevAddrResult
- RmmRttWalkNotAligned
- RmmPsmmu

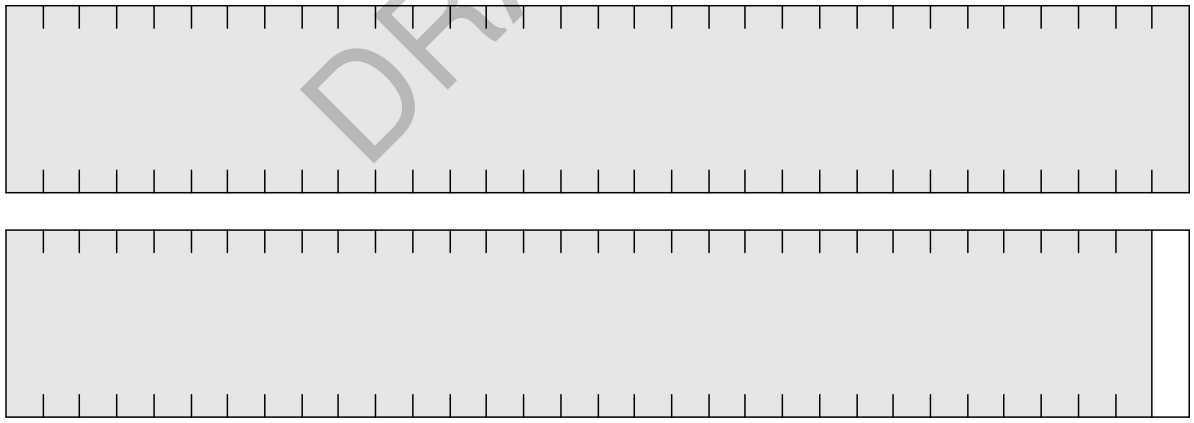
C2.3 RmmDataFlags type

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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:

- RmmVdev
- RmmPdev

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
DEV_MEM_COHERENT	A device memory location is within the system coherent memory space
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

DRAFT

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 RmmDptL0 type

The RmmDptL0 structure contains attributes of the Level 0 DPT.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The RmmDptL0 structure is an abstract type.

The members of the RmmDptL0 structure are shown in the following table.

Name	Type	Description
state	RmmDptL0State	State of the table

C2.10 RmmDptL0Entry type

The RmmDptL0Entry structure contains attributes of a Level 0 DPT entry.

The RmmDptL0Entry structure is an abstract type.

The members of the RmmDptL0Entry structure are shown in the following table.

Name	Type	Description
state	RmmDptL0EntryState	State of the entry

C2.11 RmmDptL0EntryState type

The RmmDptL0EntryState enumeration represents state of a Level 0 DPT entry.

The RmmDptL0EntryState enumeration is an abstract type.

The values of the RmmDptL0EntryState enumeration are shown in the following table.

Name	Description
DPT_L0_ENTRY_BLOCK	Level 0 DPT entry is a block entry.
DPT_L0_ENTRY_TABLE	Level 0 DPT entry is a table entry.

The RmmDptL0EntryState enumeration is used in the following types:

- RmmDptL0Entry

C2.12 RmmDptL0State type

The RmmDptL0State enumeration represents state of the Level 0 DPT.

The RmmDptL0State enumeration is an abstract type.

The values of the RmmDptL0State enumeration are shown in the following table.

Name	Description
DPT_L0_INVALID	Level 0 DPT is invalid.
DPT_L0_VALID	Level 0 DPT is valid.

The RmmDptL0State enumeration is used in the following types:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- RmmDptL0

C2.13 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	• During discovery: Feature is not supported. • During selection: Feature is not enabled.
FEATURE_TRUE	• During discovery: Feature is supported. • During selection: Feature is enabled.

The RmmFeature enumeration is used in the following types:

- RmmGlobalStatic
- RmmVdev
- RmmPdev
- RmmPsmmu
- RmmRealm

DRAFT

C2.14 RmmGlobal type

The RmmGlobal structure contains global attributes of the RMM implementation.

The RmmGlobal structure is an abstract type.

The members of the RmmGlobal structure are shown in the following table.

Name	Type	Description
static	RmmGlobalStatic	Static attributes
dynamic	RmmGlobalDynamic	Dynamic attributes

C2.15 RmmGlobalDynamic type

The RmmGlobalDynamic structure contains global dynamic attributes of the RMM implementation.

The RmmGlobalDynamic structure is an abstract type.

The members of the RmmGlobalDynamic structure are shown in the following table.

Name	Type	Description
state	RmmState	Lifecycle state

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
rmi_granule_size	UInt64	Current RMI Granule size
tracking_region_size	UInt64	Current tracking region size
num_tracked	UInt64	Number of tracking regions which have been transitioned from untracked to tracked
num_realms	UInt64	Number of Realms which have been created

The RmmGlobalDynamic structure is used in the following types:

- RmmGlobal

C2.16 RmmGlobalStatic type

The RmmGlobalStatic structure contains global static attributes of the RMM implementation.

The RmmGlobalStatic structure is an abstract type.

See also:

- Chapter A3 *Feature discovery and configuration*

The members of the RmmGlobalStatic structure are shown in the following table.

Name	Type	Description
pasz	UInt64	Physical address size in bytes
dptps	UInt64	Size of the address space covered by the DPT in bytes
l0dptsz	UInt64	Level 0 DPT size in bytes
l0gptsz	UInt64	Level 0 GPT size in bytes
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_384	RmmFeature	Whether SHA-384 is supported
feat_sha_512	RmmFeature	Whether SHA-512 is supported
feat_da	RmmFeature	Whether Realm device assignment is supported
feat_da_coh	RmmFeature	Whether coherent device assignment is supported
feat_vsmmu	RmmFeature	Whether virtual SMMU is supported

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
feat_ats	RmmFeature	Whether ATS 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
mec_count	UInt64	Number of MECs
max_recs_order	UInt64	Order of the maximum number of RECs which can be created per Realm
max_vdevs_order	UInt64	Order of the maximum number of VDEVs which can be created per PDEV
rmi_granule_size_4kb	RmmFeature	Whether 4KB RMI Granule size is supported
rmi_granule_size_16kb	RmmFeature	Whether 16KB RMI Granule size is supported
rmi_granule_size_64kb	RmmFeature	Whether 64KB RMI Granule size is supported
feat_non_tee_stream	RmmFeature	Whether NON_TEE PDEV stream type is supported
feat_vdev_krou	RmmFeature	Whether stream key refresh is required on VDEV unlock

The RmmGlobalStatic structure is used in the following types:

- RmmGlobal

DRAFT

C2.17 RmmGptL0Entry type

The RmmGptL0Entry structure contains attributes of a Level 0 GPT entry.

The RmmGptL0Entry structure is an abstract type.

The members of the RmmGptL0Entry structure are shown in the following table.

Name	Type	Description
state	RmmGptL0EntryState	State of the entry

C2.18 RmmGptL0EntryState type

The RmmGptL0EntryState enumeration represents state of a Level 0 GPT entry.

The RmmGptL0EntryState enumeration is an abstract type.

The values of the RmmGptL0EntryState enumeration are shown in the following table.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
GPT_L0_ENTRY_BLOCK	Level 0 GPT entry is a block entry.
GPT_L0_ENTRY_INTERMEDIATE	Level 0 GPT entry is is an intermediate state.
GPT_L0_ENTRY_TABLE	Level 0 GPT entry is a table entry.

The RmmGptL0EntryState enumeration is used in the following types:

- RmmGptL0Entry

C2.19 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
state	RmmGranuleState	Lifecycle state

C2.20 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
GRAN_DATA	Realm code or data.
GRAN_DELEGATED	Delegated for use by the RMM.
GRAN_DEV	Device memory, mapped into a Realm.
GRAN_INTERNAL	Used for IMPLEMENTATION DEFINED purposes.
GRAN_PDEV	Physical device object.
GRAN_RD	Realm Descriptor object
GRAN_REC	Realm Execution Context object.
GRAN_RTT	Realm Translation Table.
GRAN_UNDELEGATED	Not delegated for use by the RMM.
GRAN_VDEV	Virtual device object.
GRAN_VSMMU	Virtual SMMU object.

The RmmGranuleState enumeration is used in the following types:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- RmmGranule

C2.21 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)* [25])
HASH_SHA_384	SHA-384 (*Secure Hash Standard (SHS)* [25])
HASH_SHA_512	SHA-512 (*Secure Hash Standard (SHS)* [25])

The RmmHashAlgorithm enumeration is used in the following types:

- RmmRealm
- RmmPdev

C2.22 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_ARCH_DEV	Protected IPA which is associated with an architectural device.
HIPAS_DATA	Protected IPA which is associated with a DATA Granule.
HIPAS_MAPPED_NS	Unprotected IPA which is associated with a physical Granule.
HIPAS_NARCH_DEV	Protected IPA which is associated with a GRAN_DEV Granule.
HIPAS_UNMAPPED_NS	Unprotected IPA which is not associated with any Granule.
HIPAS_VOID	Protected IPA which is not associated with any Granule.

C2.23 RmmIrqCfg type

The RmmIrqCfg enumeration represents whether IRQ is disabled, wired type or MSI type.

The RmmIrqCfg enumeration is an abstract type.

The values of the RmmIrqCfg enumeration are shown in the following table.

Name	Description
IRQ_DISABLED	• IRQ is disabled.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
IRQ_MSI	• Interrupt is of MSI type
IRQ_WIRED	• Interrupt is of Wired type

The RmmIrqCfg enumeration is used in the following types:

- RmmPsmmu

C2.24 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

DRAFT

C2.25 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:

- B4.5.58.4 *RMI_RTT_DATA_MAP_INIT 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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

C2.26 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:

- B4.5.40.4 *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 an RMI Granule which contains REC parameters data structure. Unused bytes are zero-filled.

DRAFT

Unused bits of the RmmMeasurementDescriptorRec structure MBZ.

C2.27 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 *Private MEC*.
MEC_POLICY_SHARED	The MEC protects memory owned by multiple Realms. A MEC with this policy may be referred to as a *Shared MEC*.

The RmmMecPolicy enumeration is used in the following types:

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- RmmRealm

C2.28 RmmMemCategory type

The RmmMemCategory enumeration represents memory category.

The RmmMemCategory enumeration is an abstract type.

The values of the RmmMemCategory enumeration are shown in the following table.

Name	Description
MEM_CATEGORY_CONVENTIONAL	Conventional memory.
MEM_CATEGORY_DEV_COH	Device coherent memory.
MEM_CATEGORY_DEV_NCOH	Device non-coherent memory.

The RmmMemCategory enumeration is used in the following types:

- RmmTrackingRegion

C2.29 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

C2.30 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:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- RmmRealm

C2.31 RmmOpCanCancel type

The RmmOpCanCancel enumeration represents whether a stateful operation can be cancelled.

The RmmOpCanCancel enumeration is an abstract type.

The values of the RmmOpCanCancel enumeration are shown in the following table.

Name	Description
RMM_OP_CANNOT_CANCEL	Operation cannot be cancelled.
RMM_OP_CAN_CANCEL	Operation can be cancelled.

C2.32 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
category	RmmPdevCategory	Device category
pdev_id	Bits64	Device identifier
routing_id	Bits64	Routing identifier
rid_base	Bits32	Base of requester ID range (inclusive). For PCIe category of devices, the value is in BDF format.
rid_top	Bits32	Top of requester ID range (exclusive). For PCIe category of devices, the value is in BDF format.
spdm	RmmPdevSpdm	Whether communication with the device uses SPDM
signed_meas	RmmFeature	Whether device supports signed measurements
id_index	UInt64	Device identity index
hash_algo	RmmHashAlgorithm	Algorithm used to generate device digests
state	RmmPdevState	Lifecycle state
op	RmmPdevOperation	Operation performed on this PDEV
comm_state	RmmDevCommState	Device communication state
max_num_vdevs	UInt64	Maximum number of VDEVs which can be associated with this PDEV
num_vdevs	UInt64	Number of VDEVs associated with this PDEV
protocol_data_diges	Bits512	Protocol data digest

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

C2.33 RmmPdevCategory type

The RmmPdevCategory enumeration represents PDEV category.

The RmmPdevCategory enumeration is an abstract type.

The values of the RmmPdevCategory enumeration are shown in the following table.

Name	Description
PDEV_ENDPOINT_ACCEL_OFF_CHIP	Off-chip accelerator endpoint device
PDEV_ENDPOINT_ACCEL_ON_CHIP	On-chip accelerator endpoint device
PDEV_ROOT_PORT	Root Port

The RmmPdevCategory enumeration is used in the following types:

- RmmPdev

C2.34 RmmPdevOperation type

The RmmPdevOperation enumeration represents operation performed on a PDEV.

The RmmPdevOperation enumeration is an abstract type.

The values of the RmmPdevOperation enumeration are shown in the following table.

Name	Description
PDEV_OP_CONNECT	Request connection of a PDEV stream.
PDEV_OP_DISCONNECT	Request connection of a PDEV stream.
PDEV_OP_KEY_PURGE	Request key flush of a PDEV stream.
PDEV_OP_KEY_REFRESH	Request key refresh of a PDEV stream.
PDEV_OP_NONE	No operation.
PDEV_OP_STOP	Change state to STOPPED.
PDEV_OP_STREAM_COMPLETE	Awaiting completion of a PDEV stream operation.

The RmmPdevOperation enumeration is used in the following types:

- RmmPdev

C2.35 RmmPdevSpdm type

The RmmPdevSpdm enumeration represents whether communication with the device uses SPDM.

The RmmPdevSpdm enumeration is an abstract type.

See also:

- A9.1.2 *Device properties*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The values of the RmmPdevSpdm enumeration are shown in the following table.

Name	Description
SPDM_FALSE	Communication with the device does not use SPDM.
SPDM_TRUE	Communication with the device uses SPDM.

The RmmPdevSpdm enumeration is used in the following types:

- RmmPdev

C2.36 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_ERROR	Device has reported a fatal error.
PDEV_HAS_KEY	RMM has device public key.
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.
PDEV_STOPPED	Secure connection between the RMM and the device has been terminated.

DRAFT

The RmmPdevState enumeration is used in the following types:

- RmmPdev

C2.37 RmmPdevStream type

The RmmPdevStream structure contains attributes of a PDEV stream.

The RmmPdevStream structure is an abstract type.

The members of the RmmPdevStream structure are shown in the following table.

Name	Type	Description
handle	Bits64	Stream handle
state	RmmPdevStreamState	Stream state
stream_type	RmmPdevStreamType	Stream type
ide_sid	UInt64	IDE stream identifier

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
num_addr_range	UInt64	Number of device address ranges
addr_range	RmmAddrRange[16]	CPU physical address range used to access the device in the system memory map

The RmmPdevStream structure is used in the following types:

- RmmPdevStreamResult

C2.38 RmmPdevStreamResult type

The RmmPdevStreamResult structure contains result of looking up a PDEV stream.

The RmmPdevStreamResult structure is an abstract type.

The members of the RmmPdevStreamResult structure are shown in the following table.

Name	Type	Description
valid	RmmBoolean	Whether the lookup succeeded
stream	RmmPdevStream	Stream

DRAFT

C2.39 RmmPdevStreamState type

The RmmPdevStreamState enumeration represents the state of a PDEV stream.

The RmmPdevStreamState enumeration is an abstract type.

The values of the RmmPdevStreamState enumeration are shown in the following table.

Name	Description
PDEV_STREAM_CONNECTED	Stream is connected.
PDEV_STREAM_CONNECTING	Stream is connecting.
PDEV_STREAM_DISCONNECTED	Stream is not connected.
PDEV_STREAM_DISCONNECTING	Stream is disconnecting.
PDEV_STREAM_KEY_PURGING	Stream keys are being purged.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
PDEV_STREAM_KEY_REFRESHING	Stream keys are being refreshed.

The RmmPdevStreamState enumeration is used in the following types:

- RmmPdevStream

C2.40 RmmPdevStreamType type

The RmmPdevStreamType enumeration represents type of a PDEV stream.

The RmmPdevStreamType enumeration is an abstract type.

The values of the RmmPdevStreamType enumeration are shown in the following table.

Name	Description
PDEV_STREAM_COH	Coherent traffic between an upstream port and an accelerator endpoint device.
PDEV_STREAM_COH_SYS	Coherent traffic to an endpoint device which is protected by system construction.
PDEV_STREAM_NCOH	Non-coherent traffic between an upstream port and an endpoint device.
PDEV_STREAM_NCOH_SYS	Non-coherent traffic to an endpoint device which is protected by system construction.
PDEV_STREAM_NON_TEE	Non-TEE traffic.

DRAFT

The RmmPdevStreamType enumeration is used in the following types:

- RmmPdevStream

C2.41 RmmPsmmu type

The RmmPsmmu structure contains attributes of a PSMMU.

The RmmPsmmu structure is an abstract type.

The members of the RmmPsmmu structure are shown in the following table.

Name	Type	Description
state	RmmPsmmuState	State of the PSMMU
sid_size	UInt64	StreamID size in bits. This is read from the SMMU_IDR1.SIDSIZE register field.
irq_cfg	RmmIrqCfg	Interrupt configuration for Gerror, EventQ and PRIQ interrupts

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
cmdq_sync_irq_wired	RmmFeature	Whether the cmdq_sync interrupt is wired
gerror_intr_num	UInt32	Gerror wired interrupt number
eventq_intr_num	UInt32	EventQ wired interrupt number
priq_intr_num	UInt32	PRIQ wired interrupt number
cmdq_sync_intr_num	UInt32	CMD sync wired interrupt number
feat_ats	RmmFeature	Whether the PSMMU supports ATS
feat_pri	RmmFeature	Whether the PSMMU supports PRI
msi_config	RmmSmmuMsiConfig	MSI configuration
ppr_pending	RmmBoolean	Whether a PRI Page Request is pending on this PSMMU

C2.42 RmmPsmmuState type

The RmmPsmmuState enumeration represents the state of a PSMMU.

The RmmPsmmuState enumeration is an abstract type.

UWJWQP While the state of a PSMMU is not ACTIVE, incoming transactions from Realm security state are aborted.

See also:

- *Arm System Memory Management Unit Architecture Specification* [22]

The values of the RmmPsmmuState enumeration are shown in the following table.

Name	Description
PSMMU_ACTIVE	PSMMU has been activated.
PSMMU_INACTIVE	PSMMU has not been activated.

The RmmPsmmuState enumeration is used in the following types:

- RmmPsmmu

C2.43 RmmPsmmuStEntry type

The RmmPsmmuStEntry structure contains attributes of a PSMMU Level 1 Stream Table entry.

The RmmPsmmuStEntry structure is an abstract type.

The members of the RmmPsmmuStEntry structure are shown in the following table.

Name	Type	Description
state	RmmPsmmuStEntryState	State of the entry

The RmmPsmmuStEntry structure is used in the following types:

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- RmmPsmmuStWalkResult

C2.44 RmmPsmmuStEntryState type

The RmmPsmmuStEntryState enumeration represents state of a PSMMU Stream Table entry.

The RmmPsmmuStEntryState enumeration is an abstract type.

The values of the RmmPsmmuStEntryState enumeration are shown in the following table.

Name	Description
PSMMU_ST_ENTRY_INVALID	PSMMU Level 1 Stream Table entry is invalid.
PSMMU_ST_ENTRY_TABLE	PSMMU Level 1 Stream Table entry is a table entry.
PSMMU_ST_ENTRY_VALID	PSMMU Level 1 Stream Table entry is valid.

The RmmPsmmuStEntryState enumeration is used in the following types:

- RmmPsmmuStEntry

C2.45 RmmPsmmuStWalkResult type

The RmmPsmmuStWalkResult structure contains result of a PSMMU Stream Table walk.

The RmmPsmmuStWalkResult structure is an abstract type.

The members of the RmmPsmmuStWalkResult structure are shown in the following table.

Name	Type	Description
level	Int8	ST level reached by the walk
ste	RmmPsmmuStEntry	STE reached by the walk

C2.46 RmmReadWriteOp type

The RmmReadWriteOp enumeration represents a read or write operation.

The RmmReadWriteOp enumeration is an abstract type.

The values of the RmmReadWriteOp enumeration are shown in the following table.

Name	Description
RMM_READ	Read operation
RMM_WRITE	Write operation

C2.47 RmmRealm type

The RmmRealm structure contains attributes of a Realm.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The RmmRealm structure is an abstract type.

See also:

- A2.2 *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
rim	RmmRealmMeasurement	Realm Initial Measurement
rem	RmmRealmMeasurement[4]	Realm Extensible Measurement
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
rpv	Bits512	Realm Personalization Value
feat_da	RmmFeature	Whether Realm device assignment is enabled for this Realm
feat_ats	RmmFeature	Whether Address Translation Service is supported for devices assigned to the Realm
ats_plane	UInt64	Index of Plane whose stage 2 permissions are observed by ATS requests from devices assigned to the 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
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
num_vsmmus	UInt64	Number of VSMMUs owned by this Realm

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

C2.48 RmmRealmMeasurement type

The RmmRealmMeasurement type is realm measurement.

The RmmRealmMeasurement type is a concrete type.

The width of the RmmRealmMeasurement type is 512 bits.

C2.49 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.
REALM_ZOMBIE	Ready for destruction. Not eligible for execution.

The RmmRealmState enumeration is used in the following types:

- RmmRealm

DRAFT

C2.50 RmmRec type

The RmmRec structure contains attributes of a REC.

The RmmRec structure is an abstract type.

See also:

- A2.4 *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
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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
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 RIPAS_DESTROYED to RIPAS_RAM should be permitted
ripas_response	RmmRecResponse	Host response to RIPAS change request
dev_mem_addr	Address	Next IPA to be processed in VDEV mapping validation
dev_mem_top	Address	Top IPA of pending VDEV mapping validation
dev_mem_pa	Address	PA of device memory
dev_mem_flags	RmmDevMemFlags	VDEV mapping validation flags
dev_mem_response	RmmRecResponse	Host response to VDEV 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_1	Bits64	Virtual device ID 1
vdev_pa_1	Address	VDEV PA
vdev_id_2	Bits64	Virtual device ID 2
vdev_attest_info_1	RmmVdevAttestInfo	Attestation information for first VDEV
vdev_attest_info_2	RmmVdevAttestInfo	Attestation information for second VDEV
vsmmu_vsid	Bits64	Virtual SMMU Stream ID
vsmmu_resp	Bits64	PRI response
vsmmu_pasid	Bits64	PASID

DRAFT

C2.51 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
NO_ATTEST_IN_PROGRESS	No attestation token generation operation is in progress.

The RmmRecAttestState enumeration is used in the following types:

- RmmRec

C2.52 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

C2.53 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 elgible to run

The RmmRecFlags structure is used in the following types:

- RmmRec

C2.54 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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_COMPLETE	A VDEV request has been completed.
REC_PENDING_VDEV_REQUEST	A VDEV request is pending.
REC_PENDING_VSMMU_COMMAND	A VSMMU command is pending.
REC_PENDING_VSMMU_COMPLETE	A VSMMU command has been completed.

The RmmRecPending enumeration is used in the following types:

- RmmRec

C2.55 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
RESPONSE_ACCEPT	Host accepted the Realm request.
RESPONSE_REJECT	Host rejected the Realm request.

DRAFT

The RmmRecResponse enumeration is used in the following types:

- RmmRec

C2.56 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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

C2.57 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

C2.58 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
RIPAS_DESTROYED	Address which is inaccessible to the Realm due to an action taken by the Host.
RIPAS_DEV	Address where memory of an assigned Realm device is mapped.
RIPAS_EMPTY	Address where no Realm resources are mapped.
RIPAS_RAM	Address where private code or data owned by the Realm is mapped.

The RmmRipas enumeration is used in the following types:

- RmmRttEntry
- RmmRec

C2.59 RmmRipasChangeDestroyed type

The RmmRipasChangeDestroyed enumeration represents whether a RIPAS change from RIPAS_DESTROYED to RIPAS_RAM 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 RIPAS_DESTROYED to RIPAS_RAM should be permitted.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
NO_CHANGE_DESTROYED	A RIPAS change from RIPAS_DESTROYED to RIPAS_RAM should not be permitted.

The RmmRipasChangeDestroyed enumeration is used in the following types:

- RmmRec

C2.60 RmmRtt type

The RmmRtt structure contains an RTT.

The RmmRtt structure is an abstract type.

C2.61 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.
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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
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	Indirectly-encoded S2AP This attribute is valid if either of the following is true: • 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:

- RmmRttWalkResult

C2.62 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
RTTE_ARCH_DEV	This RTTE is identified by a Protected IPA. The output address of this RTTE points to an RMM object which is used to emulate an architectural device.
RTTE_AUX_DESTROYED	An auxiliary RTT was destroyed while a corresponding primary RTT entry was live.
RTTE_DATA	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a DATA Granule.
RTTE_MAPPED_NS	This RTTE is identified by an Unprotected IPA. The output address of this RTTE points to a Granule-aligned address within NS PAS.
RTTE_NARCH_DEV	This RTTE is identified by a Protected IPA. The output address of this RTTE points to a GRAN_DEV Granule.
RTTE_TABLE	The output address of this RTTE points to the next-level RTT.
RTTE_UNMAPPED_NS	This RTTE is identified by an Unprotected IPA. This RTTE is not associated with any Granule.
RTTE_VOID	This RTTE is identified by a Protected IPA. This RTTE is not associated with any Granule.

The RmmRttEntryState enumeration is used in the following types:

- RmmRttEntry

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

C2.63 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

C2.64 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
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:

- RmmGlobalStatic

C2.65 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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
RTT_PROTECTED	Protected IPA space.
RTT_UNPROTECTED	Unprotected IPA space.

C2.66 RmmRttS2APBase type

The RmmRttS2APBase enumeration represents S2AP base value.

The RmmRttS2APBase enumeration is an abstract type.

The values of the RmmRttS2APBase enumeration are shown in the following table.

Name	Description
S2AP_NO_ACCESS	NoAccess
S2AP_RO	RO
S2AP_RW	RW
S2AP_RW_PUX	RW+puX
S2AP_WO	WO

The RmmRttS2APBase enumeration is used in the following types:

- RmmRttS2APIndirect

DRAFT

C2.67 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

C2.68 RmmRttS2APEncoding type

The RmmRttS2APEncoding enumeration represents encoding used for S2AP.

The RmmRttS2APEncoding enumeration is an abstract type.

See also:

- A3.12 *Support for Stage 2 Access Permissions indirect encoding*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The values of the RmmRttS2APEncoding enumeration are shown in the following table.

Name	Description
S2AP_DIRECT	S2AP is encoded directly in the RTT entry.
S2AP_INDIRECT	RTT entry includes indices which indirectly specify the S2AP.

The RmmRttS2APEncoding enumeration is used in the following types:

- RmmRealm

C2.69 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	RmmRttS2APBase	Base permission index
overlay_index	UInt4	Overlay permission index

The RmmRttS2APIndirect structure is used in the following types:

- RmmRttEntry

DRAFT

C2.70 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

C2.71 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

C2.72 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

DRAFT

The RmmRttWalkResult structure is used in the following types:

- RmmRttWalkNotAligned

C2.73 RmmSmmuMsiConfig type

The RmmSmmuMsiConfig structure contains MSI configuration of an SMMU.

The RmmSmmuMsiConfig structure is an abstract type.

The members of the RmmSmmuMsiConfig structure are shown in the following table.

Name	Type	Description
gerr_addr	Address	MSI address of the GERROR interrupt (programmed to SMMU_R_GERROR_IRQ_CFG0)
gerr_data	Bits64	MSI data of the GERROR interrupt (programmed to SMMU_R_GERROR_IRQ_CFG1)
eventq_addr	Address	MSI address of the EVENTQ interrupt (programmed to SMMU_R_EVENTQ_IRQ_CFG0)
eventq_data	Bits64	MSI data of the EVENTQ interrupt (programmed to SMMU_R_EVENTQ_IRQ_CFG1)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
priq_addr	Address	MSI address of the PRIQ interrupt (programmed to SMMU_R_PRIQ_IRQ_CFG0)
priq_data	Bits64	MSI data of the PRIQ interrupt (programmed to SMMU_R_PRIQ_IRQ_CFG1)

The RmmSmmuMsiConfig structure is used in the following types:

- RmmPsmmu
- RmmVsmmu

C2.74 RmmState type

The RmmState enumeration represents lifecycle state of the RMM.

The RmmState enumeration is an abstract type.

The values of the RmmState enumeration are shown in the following table.

Name	Description
RMM_STATE_ACTIVE	RMM is active.
RMM_STATE_INIT	Initial state of the RMM.

The RmmState enumeration is used in the following types:

- RmmGlobalDynamic

DRAFT

C2.75 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

C2.76 RmmTrackingRegion type

The RmmTrackingRegion structure contains attributes of a Granule tracking region.

The RmmTrackingRegion structure is an abstract type.

The members of the RmmTrackingRegion structure are shown in the following table.

Name	Type	Description
state	RmmTrackingRegionState	Tracking region state.
category	RmmMemCategory	Memory category.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

C2.77 RmmTrackingRegionState type

The RmmTrackingRegionState enumeration represents tracking region state.

The RmmTrackingRegionState enumeration is an abstract type.

The values of the RmmTrackingRegionState enumeration are shown in the following table.

Name	Description
TRACKING_COARSE	Region is tracked at the granularity of the Tracking Region size.
TRACKING_FINE	Region is tracked at the granularity of the RMI Granule size.
TRACKING_NONE	Region is not tracked.
TRACKING_RESERVED	Region is reserved for use by the platform.

The RmmTrackingRegionState enumeration is used in the following types:

- RmmTrackingRegion

C2.78 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
tdi_id	Bits64	TDI identifier
pdev	Address	PA of parent PDEV
realm	Address	PA of RD of Realm which owns this VDEV
vdev_state	RmmVdevState	VDEV lifecycle state
dma_state	RmmVdevDmaState	DMA state
non_ats_plane	UInt64	Index of Plane whose stage 2 permissions are observed by non-ATS requests from the device
op	RmmVdevOperation	Operation performed on this VDEV
comm_state	RmmDevCommState	Device communication state
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.
attest_info	RmmVdevAttestInfo	Attestation information
meas_digest	Bits512	Measurement digest

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Type	Description
report_digest	Bits512	Interface report digest
num_addr_range	UInt64	Number of device address ranges
addr_range	RmmAddrRange[8]	CPU physical address range used to access the device in the system memory map These ranges include both coherent and non- coherent device memory.

C2.79 RmmVdevAddrResult type

The RmmVdevAddrResult structure contains result of scanning VDEV address ranges.

The RmmVdevAddrResult structure is an abstract type.

The members of the RmmVdevAddrResult structure are shown in the following table.

Name	Type	Description
valid	RmmBoolean	Whether a mapped Granule was found
addr	Address	PA of mapped Granule

DRAFT

C2.80 RmmVdevAttestInfo type

The RmmVdevAttestInfo structure contains attestation information for a VDEV.

The RmmVdevAttestInfo structure is an abstract type.

The members of the RmmVdevAttestInfo structure are shown in the following table.

Name	Type	Description
lock_nonce	UInt64	Nonce generated on most recent transition to LOCKED state
meas_nonce	UInt64	Nonce generated on most recent GET_MEASUREMENT request
report_nonce	UInt64	Nonce generated on most recent GET_INTERFACE_REPORT request

The RmmVdevAttestInfo structure is used in the following types:

- RmmVdev
- RmmRec

C2.81 RmmVdevDmaState type

The RmmVdevDmaState enumeration represents the state of DMA for a VDEV.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

The RmmVdevDmaState enumeration is an abstract type.

The values of the RmmVdevDmaState enumeration are shown in the following table.

Name	Description
VDEV_DMA_DISABLED	DMA is disabled.
VDEV_DMA_ENABLED	DMA is enabled.

The RmmVdevDmaState enumeration is used in the following types:

- RmmVdev

C2.82 RmmVdevOperation type

The RmmVdevOperation enumeration represents operation performed on a VDEV.

The RmmVdevOperation enumeration is an abstract type.

The values of the RmmVdevOperation enumeration are shown in the following table.

Name	Description
VDEV_OP_GET_MEAS	Request a measurement report.
VDEV_OP_GET_REPORT	Request an interface report.
VDEV_OP_KEY_PURGE	Purge keys of PDEV streams.
VDEV_OP_KEY_REFRESH	Refresh keys of PDEV streams.
VDEV_OP_LOCK	Change state to LOCKED.
VDEV_OP_NONE	No operation.
VDEV_OP_START	Change state to STARTED.
VDEV_OP_UNLOCK	Change state to UNLOCKED.

DRAFT

The RmmVdevOperation enumeration is used in the following types:

- RmmVdev

C2.83 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_ERROR	Device interface has reported a fatal error.
VDEV_KEY_PURGE	Waiting for purge of inactive keys to be performed on all streams associated with the device interface.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Name	Description
VDEV_KEY_REFRESH	Waiting for key refresh to be performed on all streams associated with the device interface.
VDEV_LOCKED	Device interface is locked.
VDEV_NEW	Initial state of the device interface.
VDEV_STARTED	Device interface is started.
VDEV_UNLOCKED	Device interface is unlocked.

The RmmVdevState enumeration is used in the following types:

- RmmVdev

C2.84 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
state	RmmVsmmuState	State of the VSMMU
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
msi_config	RmmSmmuMsiConfig	MSI configuration

DRAFT

C2.85 RmmVsmmuState type

The RmmVsmmuState enumeration represents the state of a VSMMU.

The RmmVsmmuState enumeration is an abstract type.

The values of the RmmVsmmuState enumeration are shown in the following table.

Name	Description
VSMMU_ACTIVE	VSMMU has been activated by the Realm.
VSMMU_INACTIVE	VSMMU has not been activated by the Realm.

The RmmVsmmuState enumeration is used in the following types:

- RmmVsmmu

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter C3
Generic types

DRAFT

This section defines types which are shared between RMM interfaces and descriptions of RMM abstract state.

See also:

- B4.6 *RMI types*
- B5.5 *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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

DRAFT

Part D
Usage

Chapter D1
Flows

DRAFT

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_RANGE_DELEGATE and RMI_GRANULE_RANGE_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 GRAN_DELEGATED state, from which RMM data structures and Realm data can be allocated on demand.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

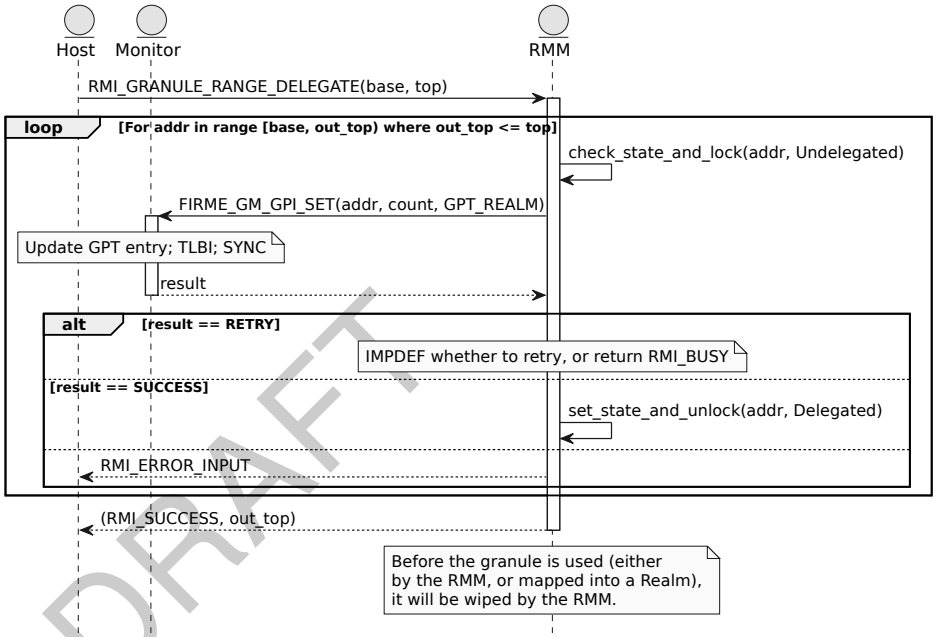
D1.1 Granule delegation flows

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.



DRAFT

Figure D1.1: Granule delegation flow

See also:

- A2.3.6 *Granule state*
- B4.5.8 *RMI_GRANULE_RANGE_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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

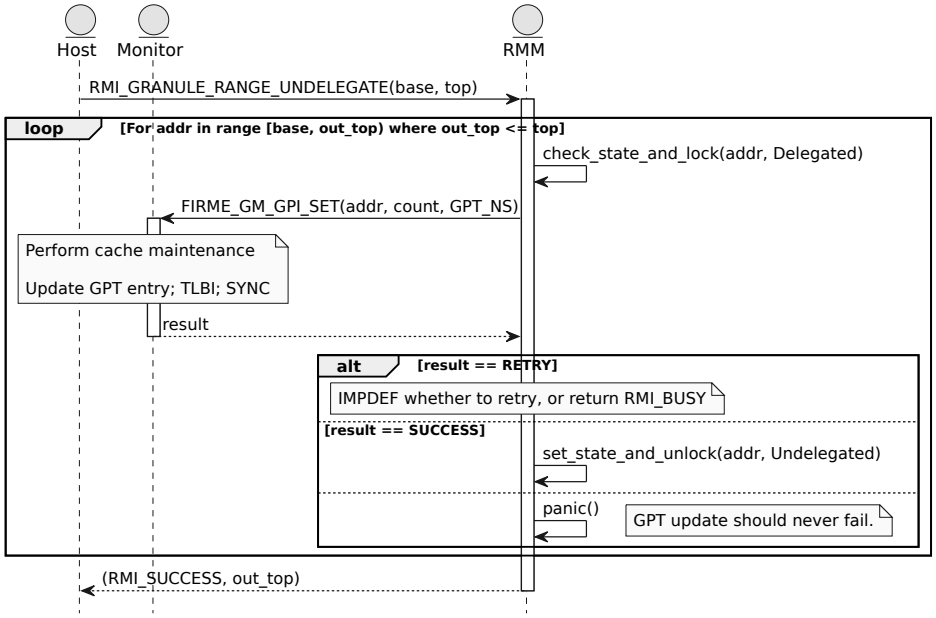


Figure D1.2: Granule undelegation flow

DRAFT

See also:

- A2.3.6 *Granule state*
- B4.5.9 *RMI_GRANULE_RANGE_UNDELEGATE command*
- D1.1.1 *Granule delegation flow*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

D1.2 Realm lifecycle flows

This section contains flows which relate to the Realm lifecycle.

See also:

- A2.2.5 *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.

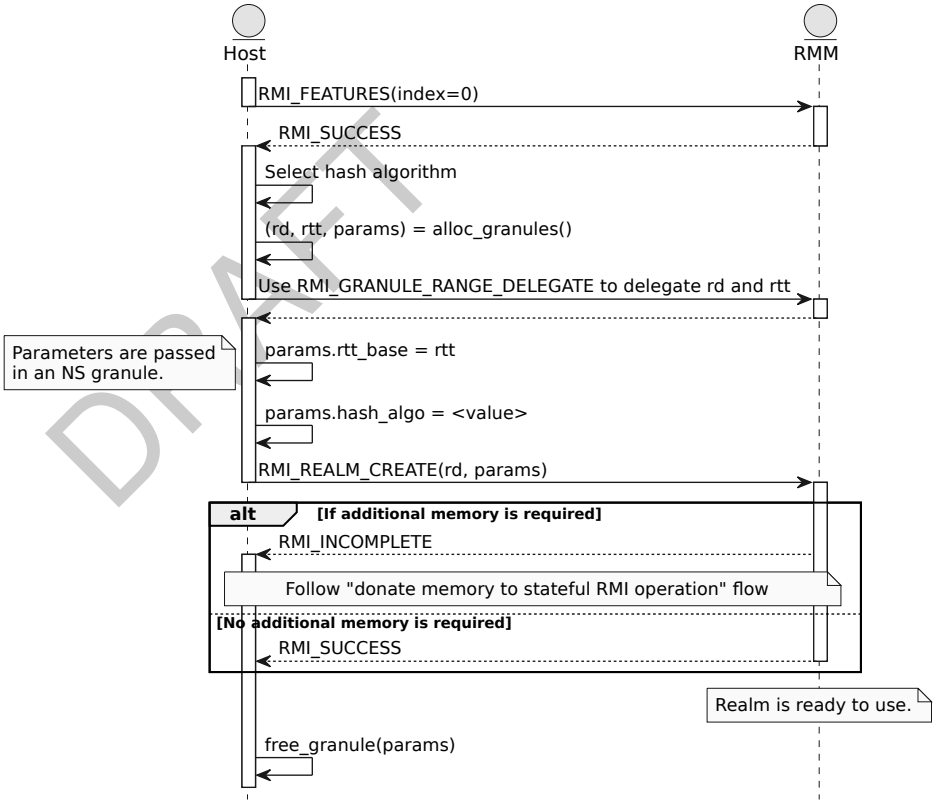


Figure D1.3: Realm creation flow

See also:

- B4.3.4 *Object creation and destruction*
- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*
- B4.5.37 *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).

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

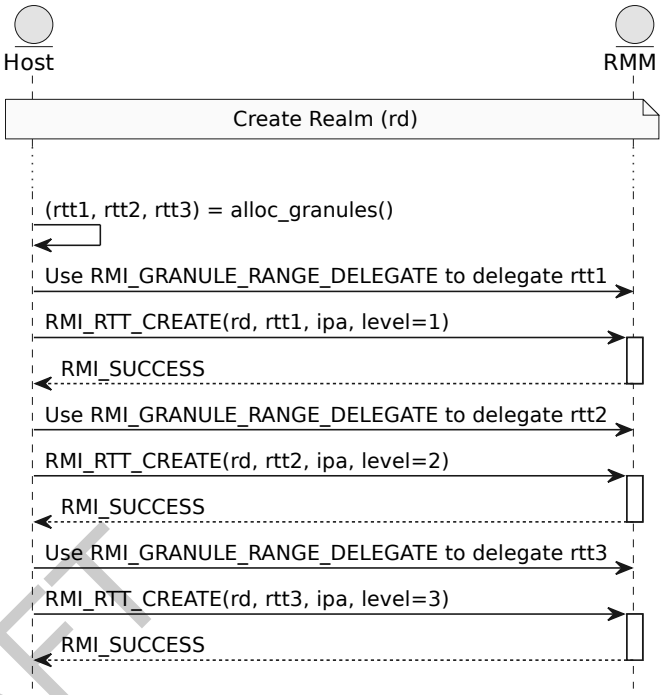


Figure D1.4: RTT creation flow

DRAFT

See also:

- Chapter A5 *Realm memory management*
- B4.5.56 *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 RIPAS_EMPTY. There are two ways in which the Host can set the RIPAS of a given page of Protected IPA space to RIPAS_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_RTT_DATA_MAP_INIT. 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

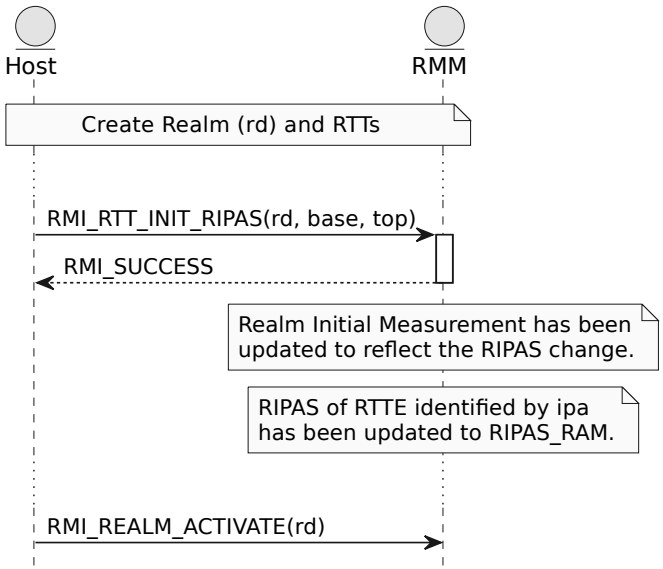


Figure D1.5: RIPAS initialization flow

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

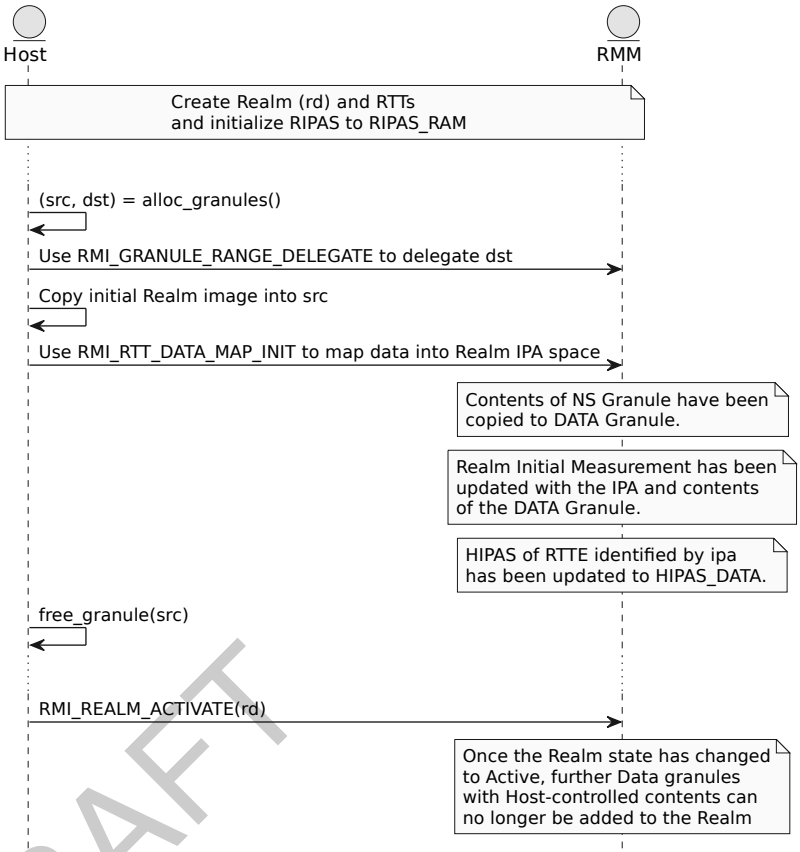


Figure D1.6: Realm data creation flow

DRAFT

See also:

- A2.3.6 *Granule state*
- A5.2.2 *Realm IPA state*
- A7.1.1 *Realm Initial Measurement*
- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*
- B4.5.58 *RMI_RTT_DATA_MAP_INIT command*
- B4.5.65 *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.

Once the REC has been created, the `params` Granule can be reallocated by the Host.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

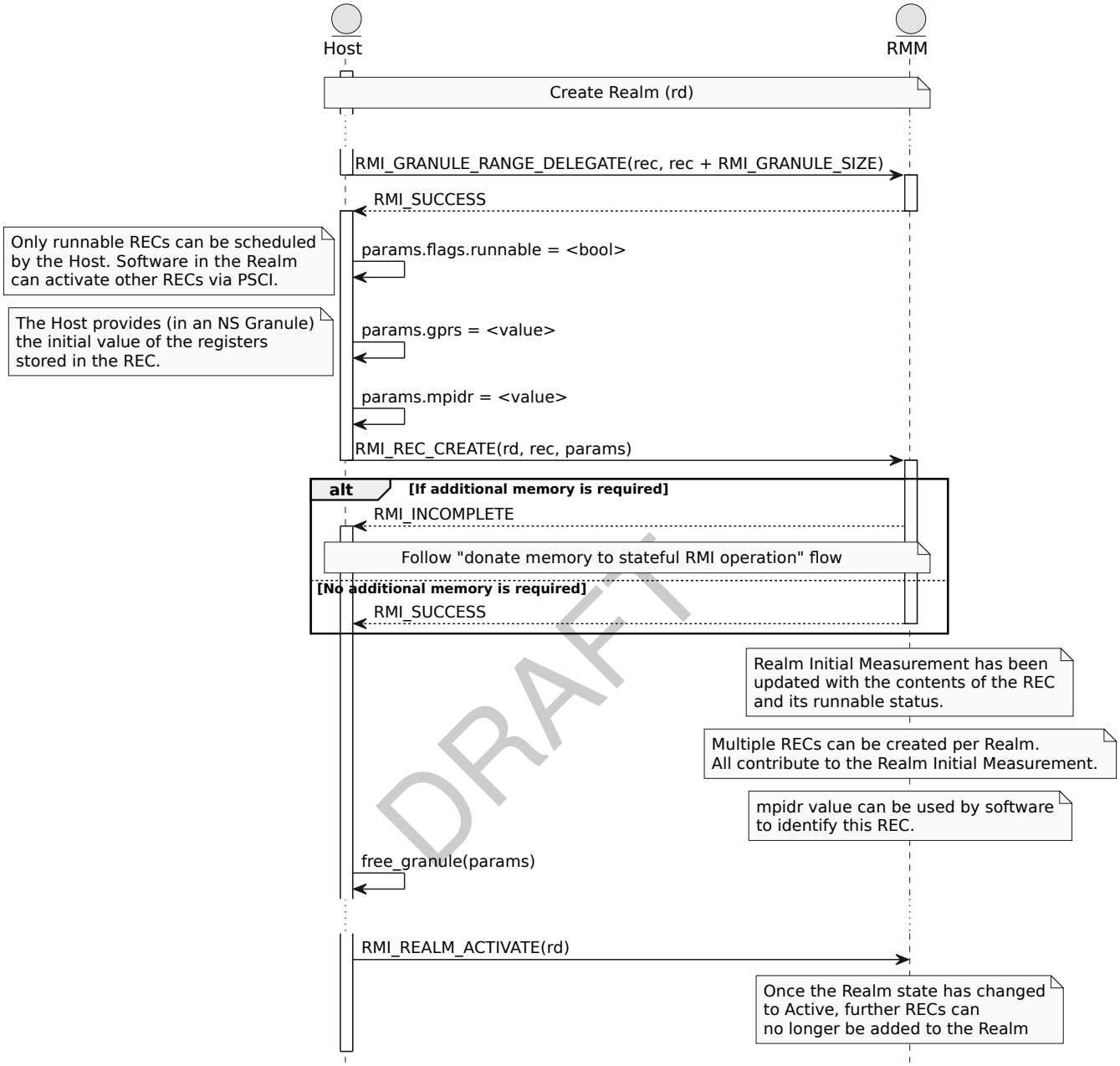


Figure D1.7: REC creation flow

See also:

- B4.3.4 *Object creation and destruction*
- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*
- B4.5.40 *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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

To destroy a Realm, the Host must perform the following actions:

1a. Ensure that no REC owned by the Realm is running.

1b. Terminate the Realm, which causes it to enter REALM_ZOMBIE state. Realm termination can result in an intermediate state, therefore requiring multiple calls to complete the destruction. On entry to REALM_ZOMBIE state, the RMM ensures that all DMA initiated by the Realm has stopped.

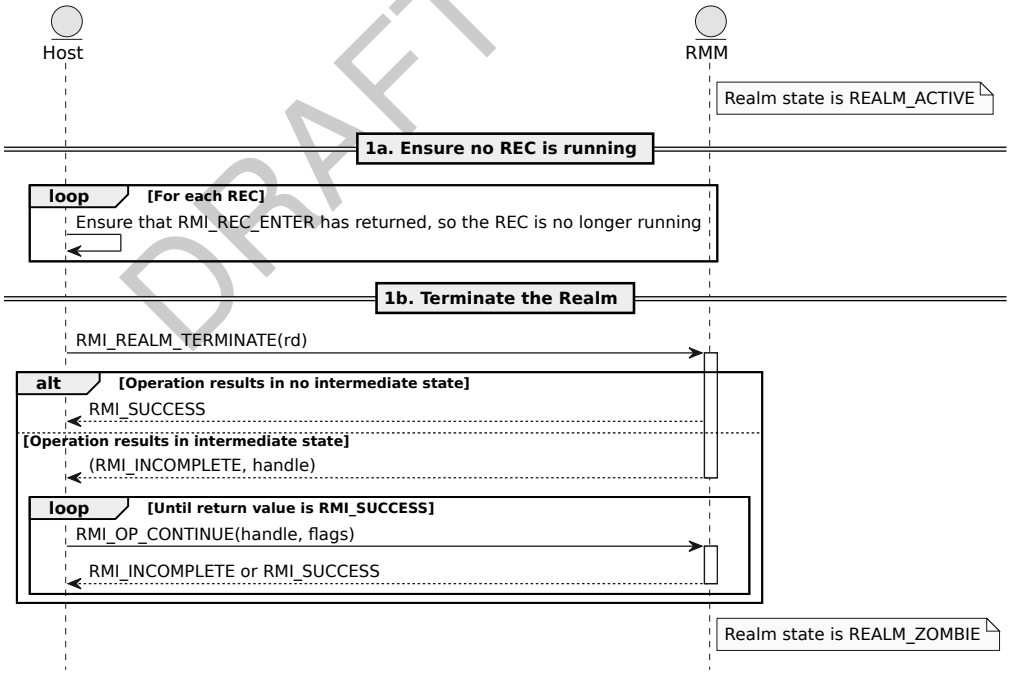
2. Make the Realm non-live. This is done by destroying (in any order) the objects which are associated with the Realm:

- Data Granules
- REC
- RTT
- VDEV
- VSMMU

3. Destroy the Realm. This is an operation which can result in an intermediate state, therefore requiring multiple calls to complete the destruction.

Steps 1 and 2 above can be performed in either order.

Once each object (DATA, REC, RTT and RD) has been destroyed, the corresponding Granules can be undelegated and reallocated by the Host.



DRAFT

Figure D1.8: Realm destruction flow (part 1 of 2)

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

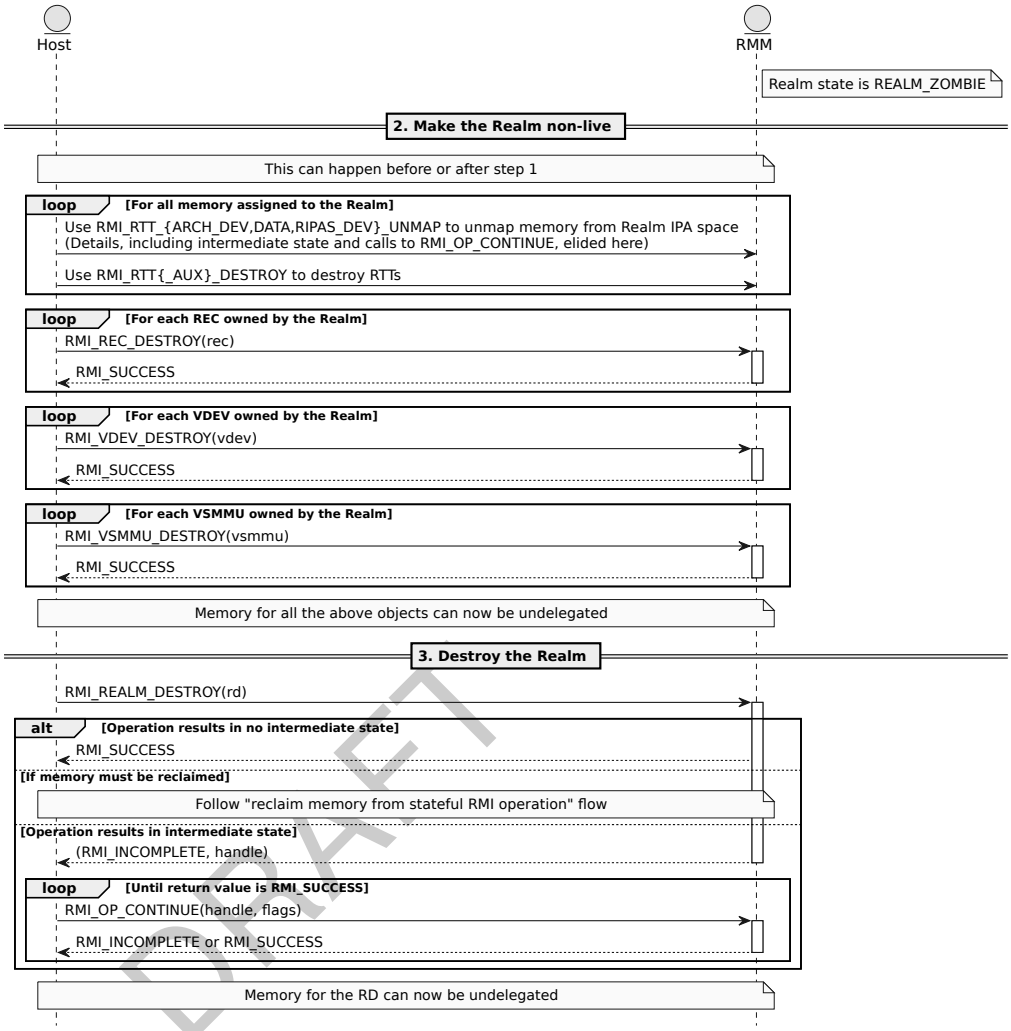


Figure D1.9: Realm destruction flow (part 2 of 2)

See also:

- A2.2.4 *Realm liveness*
- B4.3.2 *Stateful RMI operations*
- B4.3.4 *Object creation and destruction*
- B4.5.9 *RMI_GRANULE_RANGE_UNDELEGATE command*
- B4.5.38 *RMI_REALM_DESTROY command*
- B4.5.39 *RMI_REALM_TERMINATE command*
- B4.5.41 *RMI_REC_DESTROY command*
- B4.5.59 *RMI_RTT_DATA_UNMAP command*
- B4.5.60 *RMI_RTT_DESTROY command*
- B4.5.75 *RMI_VDEV_DESTROY command*
- B4.5.86 *RMI_VSMMU_DESTROY command*
- D1.2.1 *Realm creation flow*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

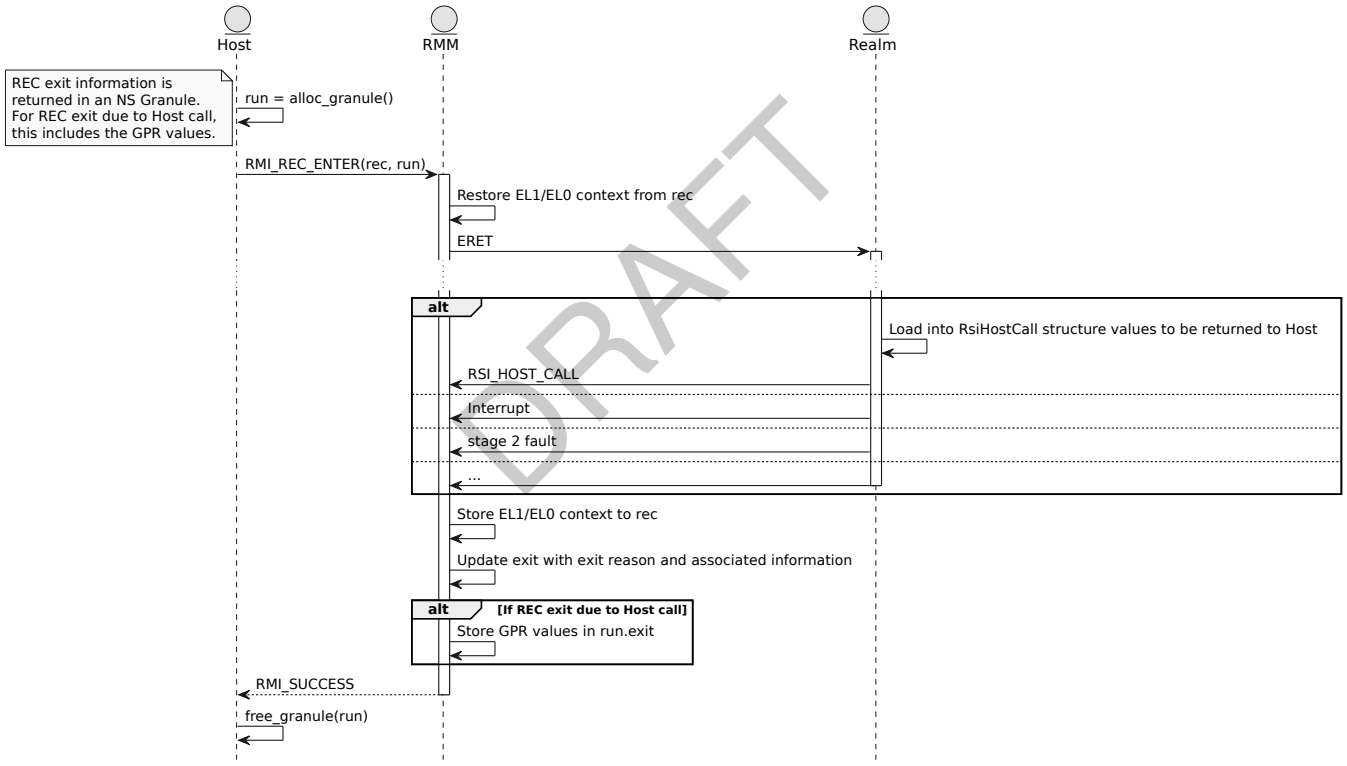


Figure D1.10: Realm entry and exit flow

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

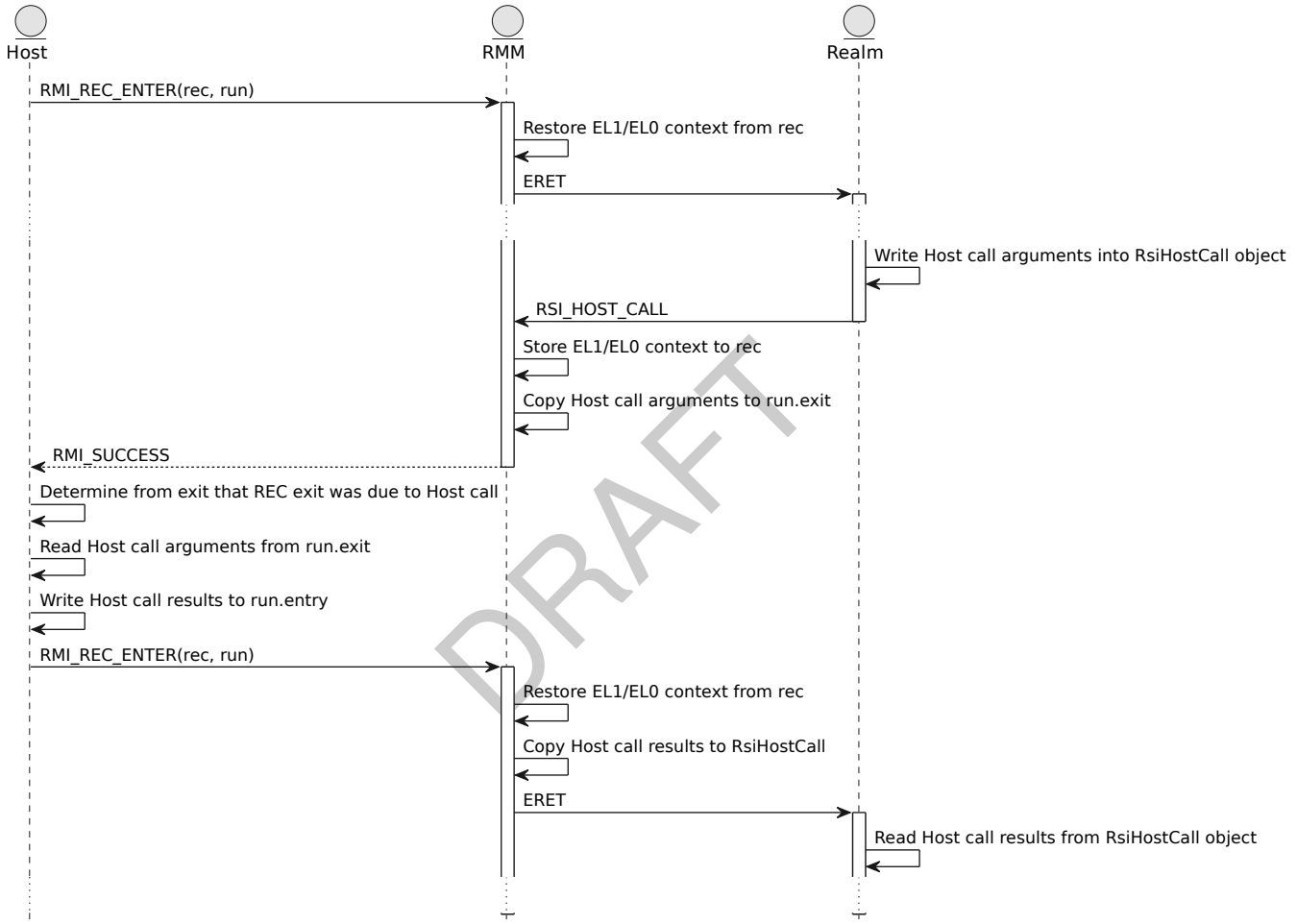


Figure D1.11: Host call flow

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

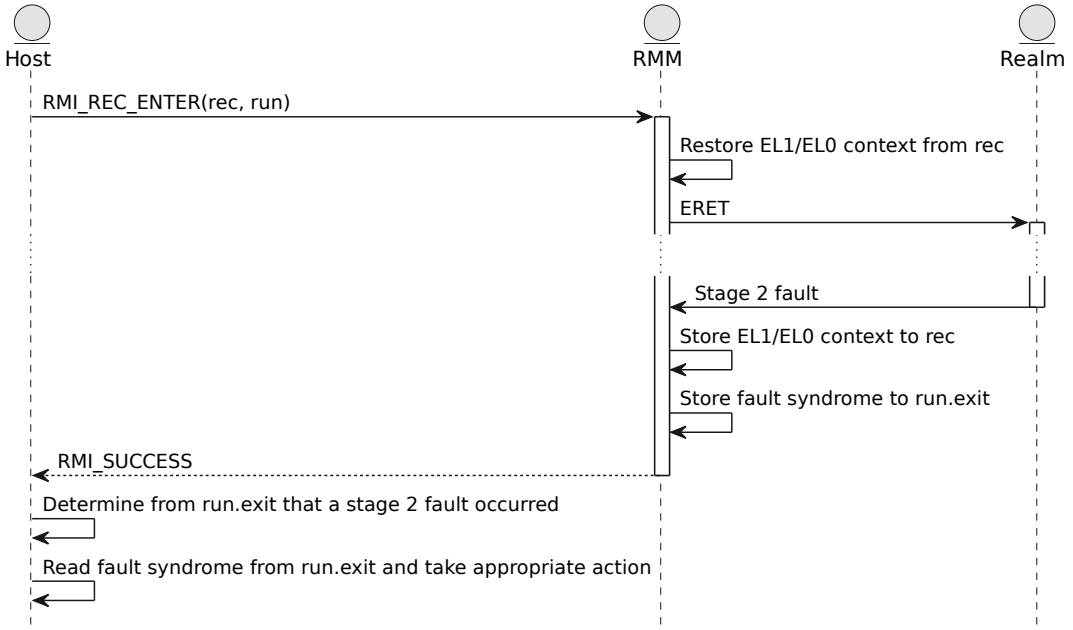


Figure D1.12: REC exit due to Data Abort flow

DRAFT

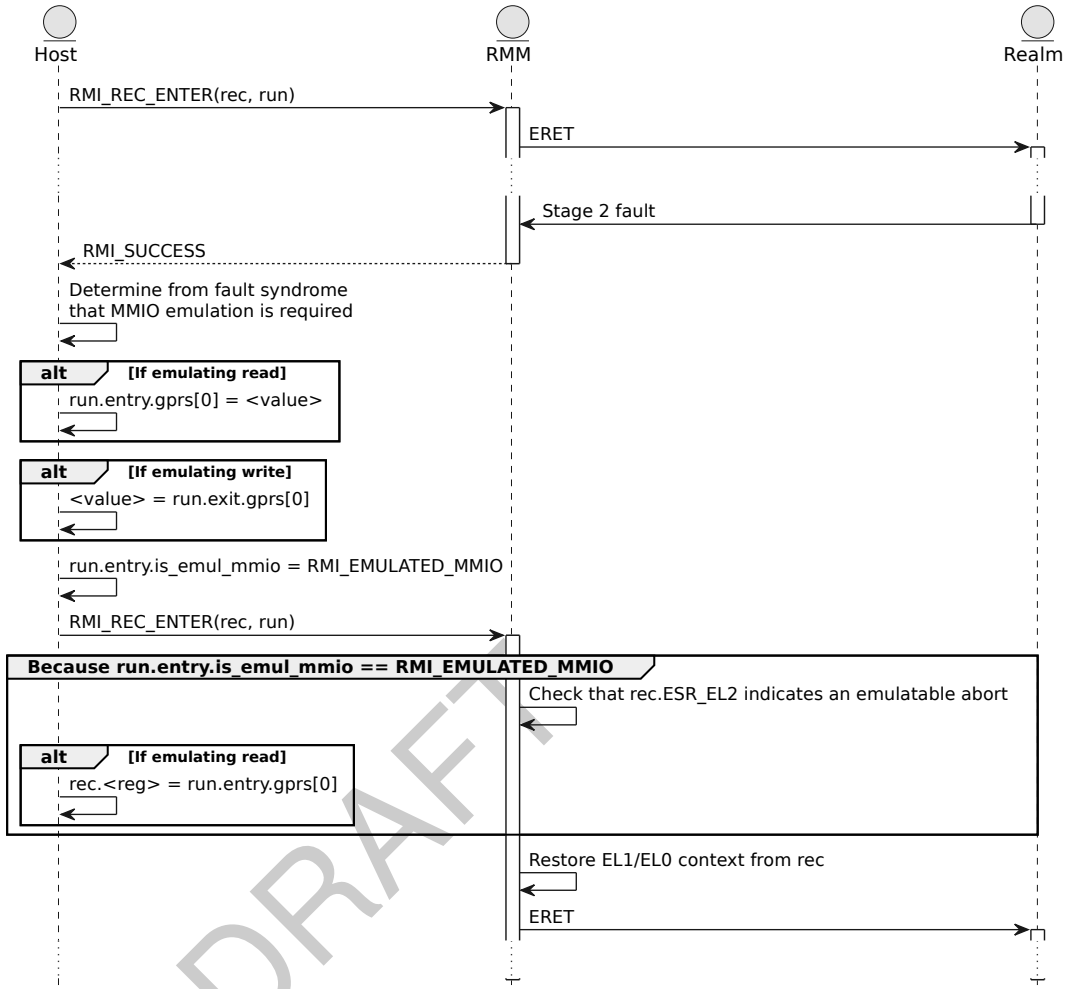
See also:

- 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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential



DRAFT

Figure D1.13: MMIO emulation flow

See also:

- Chapter A4 *Realm exception model*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

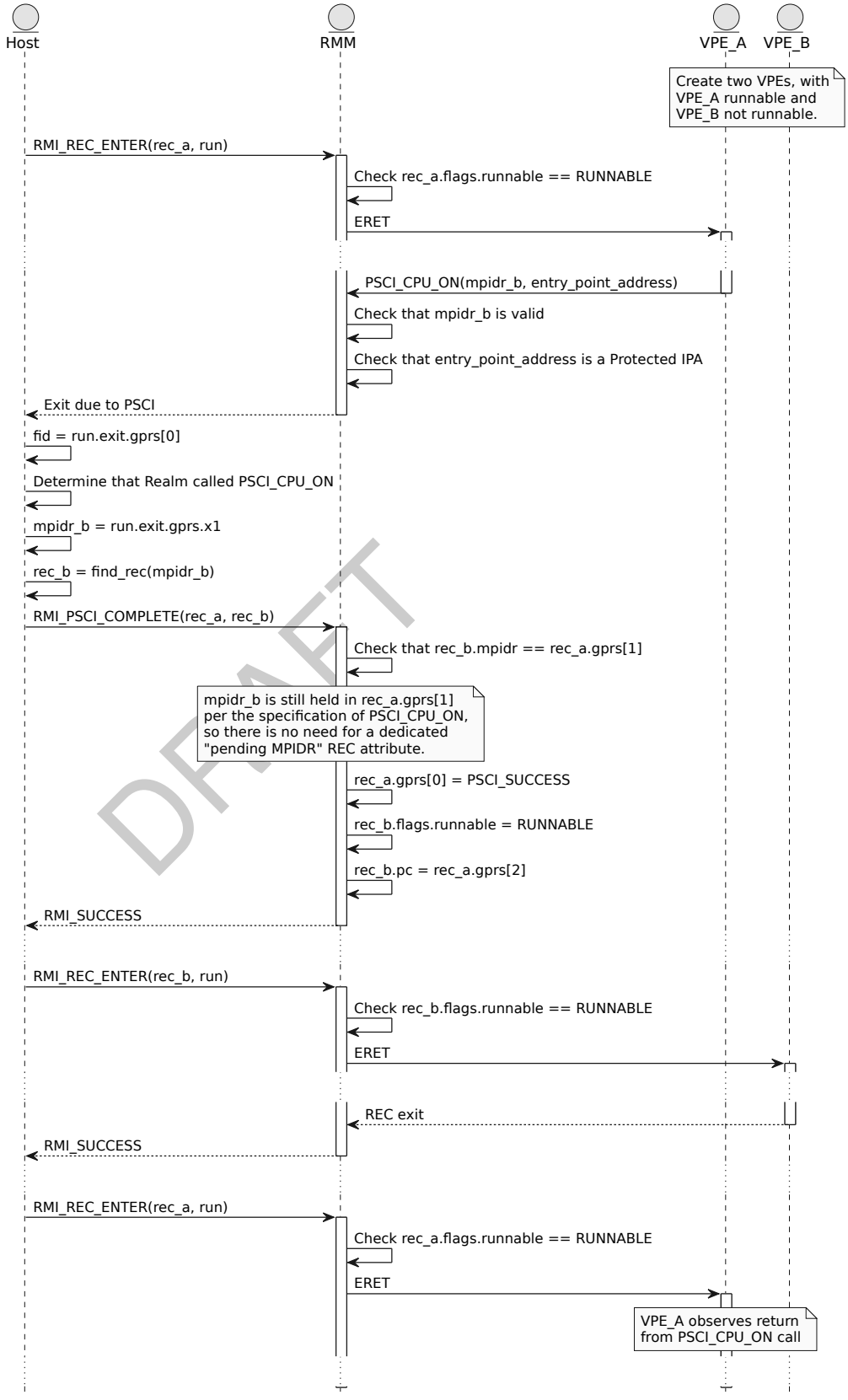


Figure D1.14: PSCI_CPU_ON flow

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

See also:

- B4.5.28 *RMI_PSCI_COMPLETE command*
- B6.3.3 *PSCI_CPU_ON command*

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

D1.5 Realm memory management flows

This section contains flows which relate to management of Realm memory.

See also:

- Chapter A5 *Realm memory management*

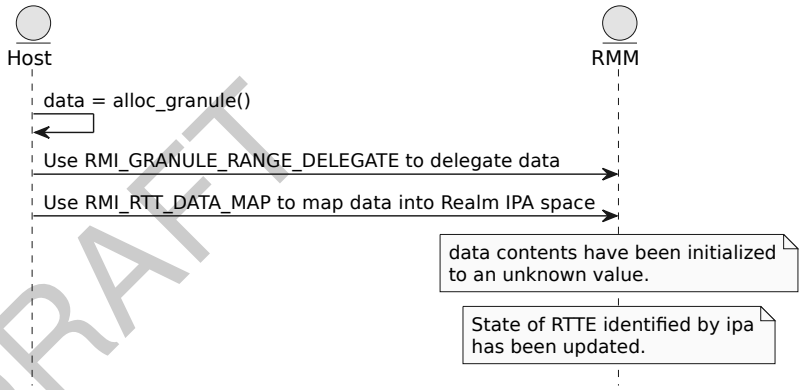
D1.5.1 Add memory to Active Realm flow

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.



DRAFT

Figure D1.15: Add memory to active Realm flow

See also:

- A2.2.5 *Realm lifecycle*
- Chapter A5 *Realm memory management*
- B4.5.8 *RMI_GRANULE_RANGE_DELEGATE command*
- B4.5.57 *RMI_RTT_DATA_MAP command*

D1.5.2 NS memory flow

The following diagram describes how NS memory can be mapped into a Realm.

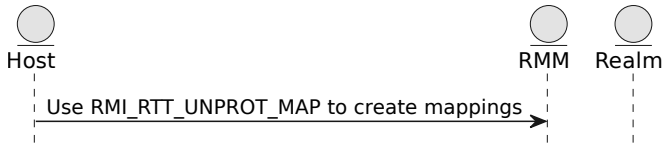


Figure D1.16: NS memory mapping flow

See also:

- Chapter A5 *Realm memory management*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

- B4.5.69 *RMI_RTT_UNPROT_MAP command*
- B4.5.70 *RMI_RTT_UNPROT_UNMAP 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.

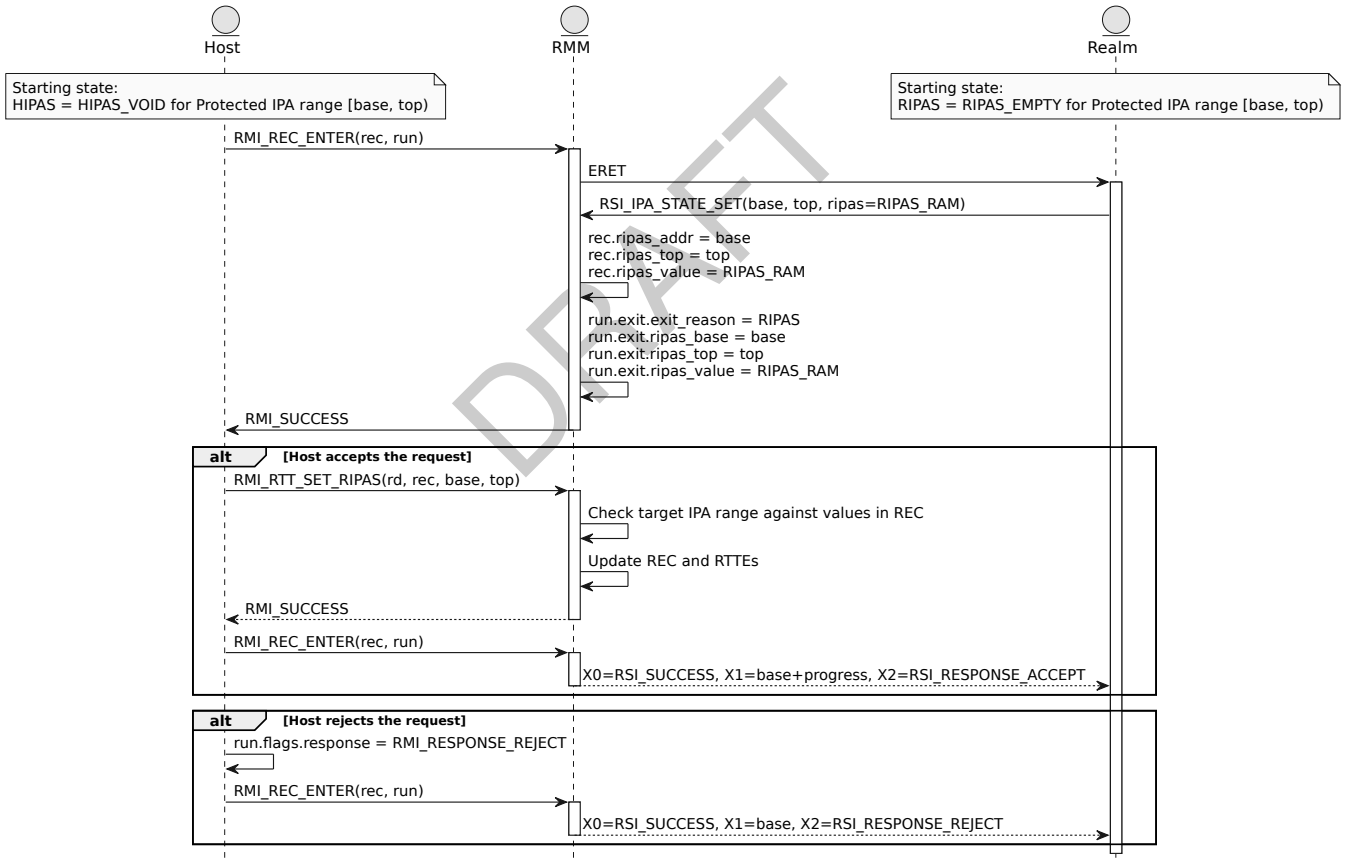


Figure D1.17: RIPAS change flow

See also:

- A5.4 *RIPAS change*
- B4.5.42 *RMI_REC_ENTER command*
- B4.5.67 *RMI_RTT_SET_RIPAS command*
- B5.4.7 *RSI_IPA_STATE_SET command*
- D2.2 *Realm shared memory protocol flow*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

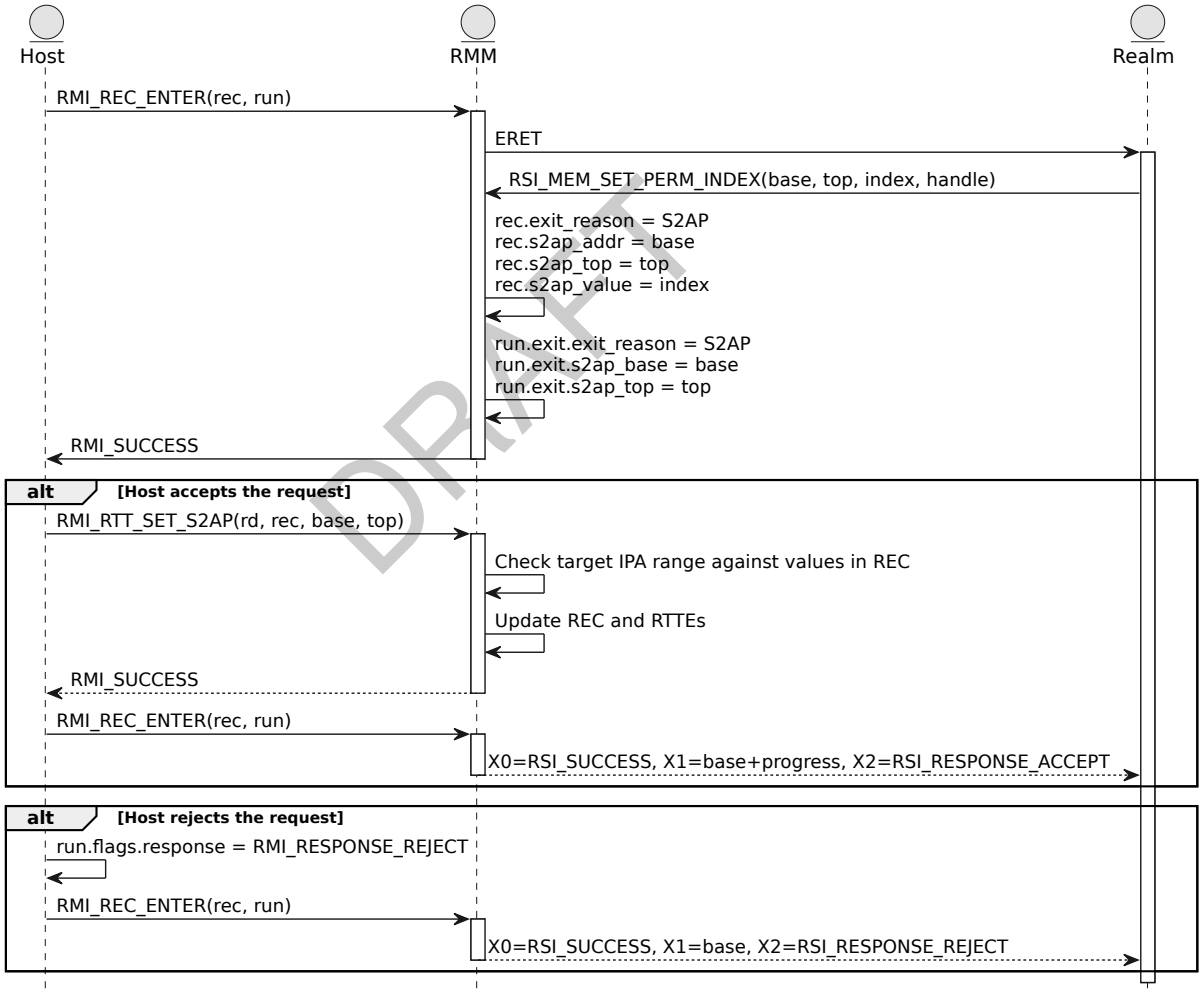


Figure D1.18: S2AP change flow

See also:

- A10.3.2.3 *Stage 2 Access Permissions change within a multi-Plane Realm*
- B4.5.42 *RMI_REC_ENTER command*
- B4.5.68 *RMI_RTT_SET_S2AP command*
- B5.4.11 *RSI_MEM_SET_PERM_INDEX command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

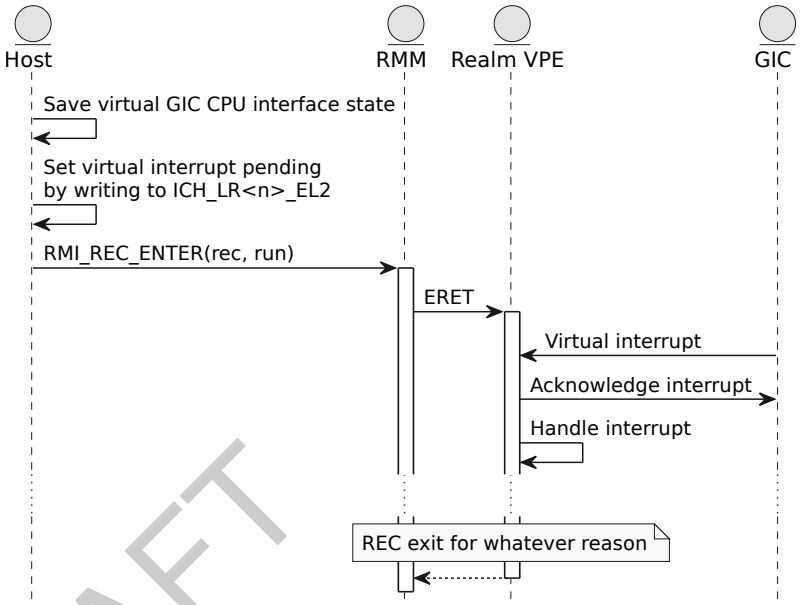


Figure D1.19: Interrupt injection flow

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

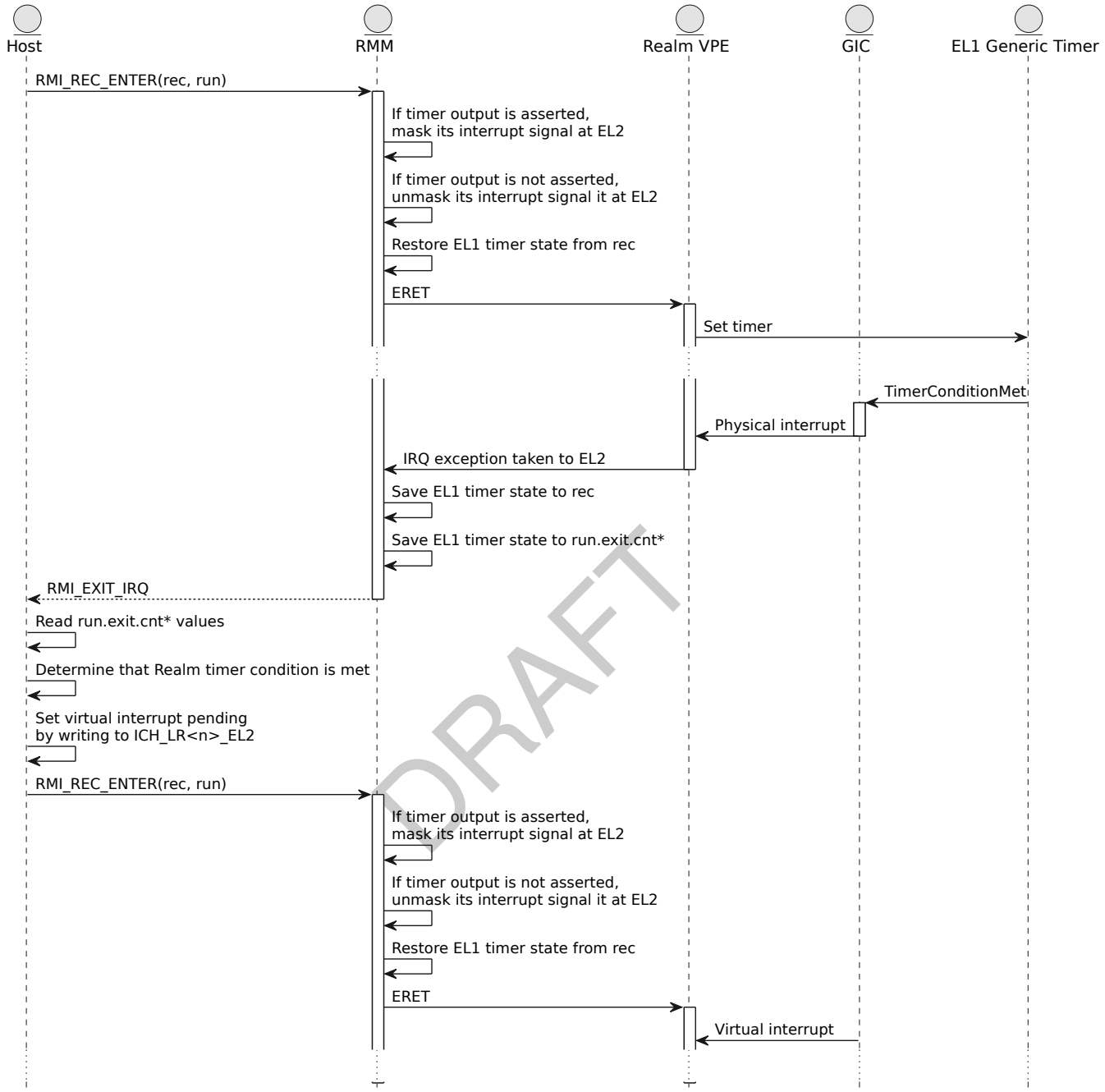


Figure D1.20: Timer interrupt delivery flow

See also:

- A6.2 *Realm timers*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

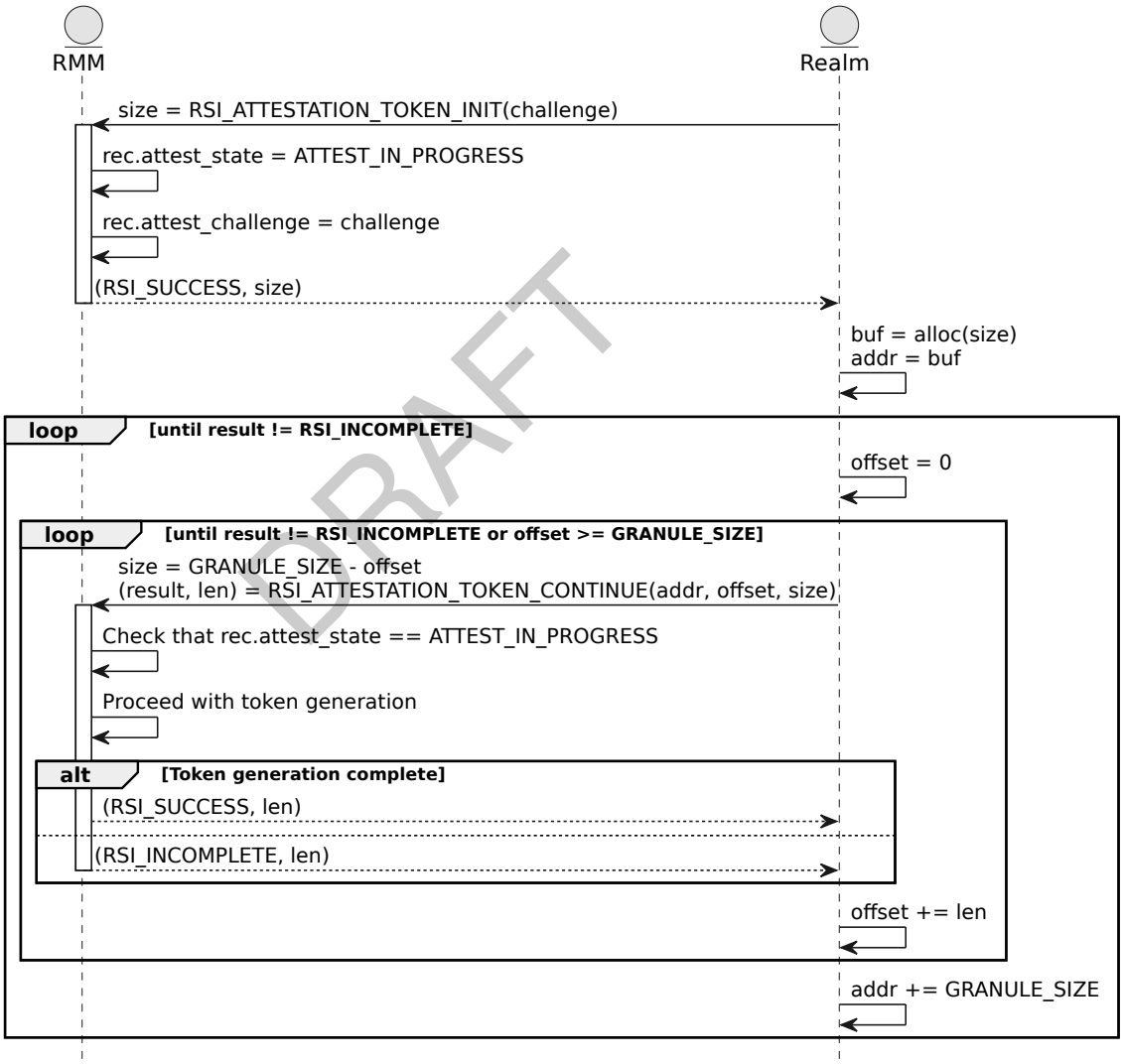


Figure D1.21: Attestation token generation flow

See also:

- A7.2.2 *Attestation token generation*
- B5.4.2 *RSI_ATTESTATION_TOKEN_CONTINUE command*
- B5.4.3 *RSI_ATTESTATION_TOKEN_INIT command*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

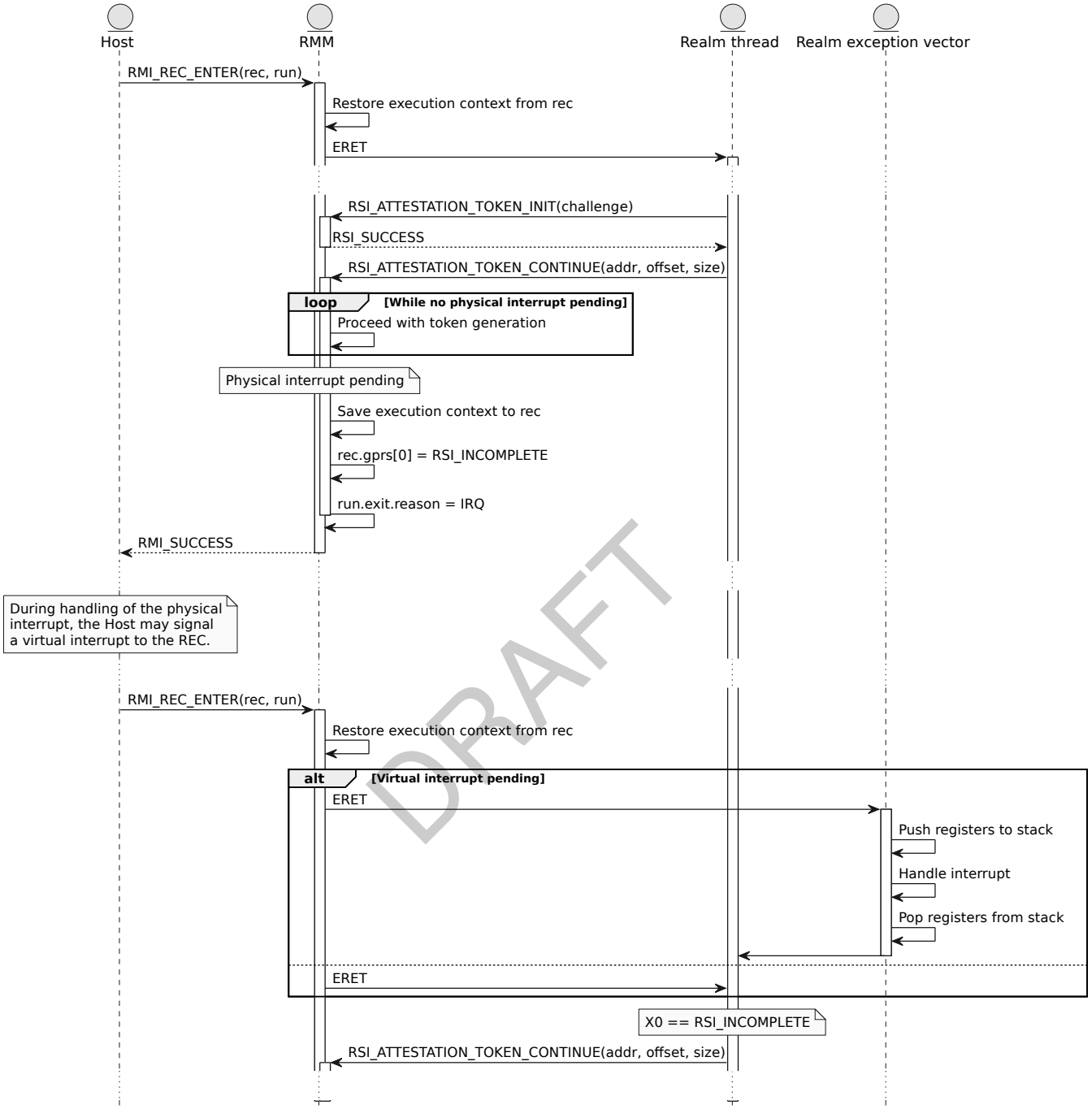


Figure D1.22: Attestation token generation with interrupt handling flow

See also:

- A4.3.5 *REC exit due to IRQ*
- A6.1 *Realm interrupts*
- A7.2.2 *Attestation token generation*
- B5.4.2 *RSI_ATTESTATION_TOKEN_CONTINUE command*
- B5.4.3 *RSI_ATTESTATION_TOKEN_INIT command*
- D1.3.1 *Realm entry and exit flow*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

D1.8 Realm device assignment flows

See Chapter A9 *Realm device assignment*.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Chapter D2
Realm shared memory protocol

DRAFT

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*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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 RIPAS_RAM, the page is private.
- If the RIPAS of the Protected IPA is RIPAS_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.

S_{ZBPXF} For device access to shared memory which is mapped in a Realm's IPA space, an Unprotected IPA should be used as the DMA address.

DRAFT

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.

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

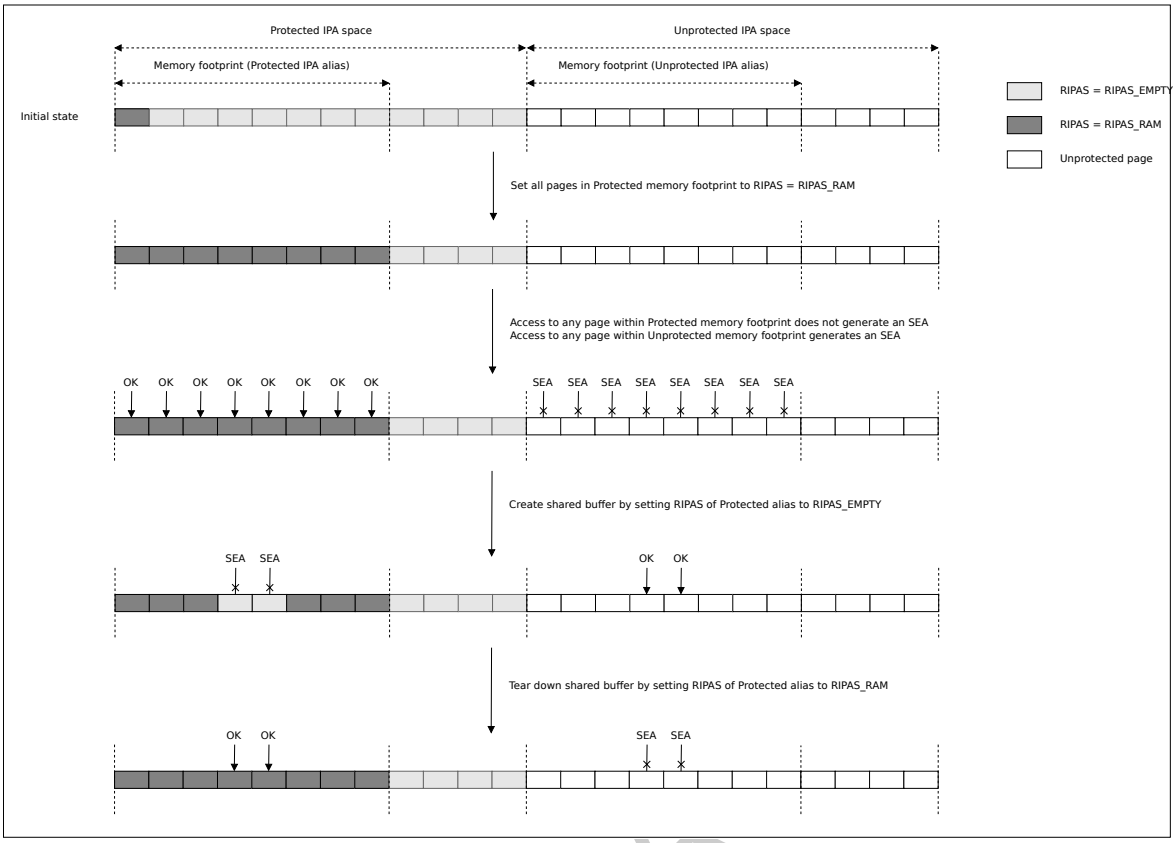


Figure D2.1: Realm shared memory protocol flow

DRA

See also:

- D1.5.3 *RIPAS change flow*

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Glossary

ASL

Arm Specification Language
Language used to express pseudocode implementations. Formal language definition can be found in *Arm Specification Language Reference Manual* [24].

ATC

Address Translation Cache

ATS

Address Translation Service

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

CPAK

CCA Platform Attestation Key Key used to sign the CCA platform attestation token.

CXL

Compute eXpress Link

CXL TSP

CXL Trusted Execution Environment Security Protocol

DOE

Data Object Exchange See *PCI Express 6.0 specification* [16]

DPT

Device Permission Table

DSM

Device Security Manager See *PCI Express 6.0 specification* [16]

EAT

Entity Attestation Token

ECAM

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Enhanced Configuration Access Mechanism See *PCI Express 6.0 specification* [16]

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.

HDM

Host-managed Device memory

HDM-H

Host-managed Device memory (Host-coherent)

HIPAS

Host IPA state

Host

Software executing in Non-secure Security state which manages resources used by Realms

IDE

Integrity and Data Encryption
See *PCI Express 6.0 specification* [16]

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

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

PPR

PRI Page Request

PRG

Page Request Group

PRI

Page Request Interface

PSCI

Power State Control Interface
See *Arm Power State Coordination Interface (PSCI)* [28]

PSMMU

Physical System Memory Management Unit
See *Arm System Memory Management Unit Architecture Specification* [22]

RAK

Realm Attestation Key Key used to sign the Realm attestation token.

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

RD

Realm Descriptor
Object which stores attributes of a Realm.

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

RHI

Realm Host Interface

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

RMSD

Realm Management Security Domain

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

DRAFT

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential

Should Be Zero

SEA

Synchronous External Abort

SGI

Software Generated Interrupt

SMCCC

SMC Calling Convention
See *Arm SMC Calling Convention* [23]

SMMU

System Memory Management Unit
See *Arm System Memory Management Unit Architecture Specification* [22]

SPDM

Security Protocol and Data Model
See *Security Protocol and Data Model (SPDM)* [21] and *Secured Messages using SPDM Specification version 1.1.0* [17]

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* [22]

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

Copyright © 2022-2026 Arm Limited or its affiliates. All rights reserved.
Non-confidential