

Project use only

RaSTA Protocol Reference Stack

Attachment of Software Component Design Specification
SIL4

Project:	RaSTA Protocol Reference Stack	Customer:	SBB
Author:	N. Andres	Version:	V1.1.0
Document-Id:	SBB-RaSTA-071	Status:	Release
Date:	20/12/2022	Filename:	SBB-RaSTA-071- AttachmentOfSoftwareComponentDesignSpecification- SIL4.pdf

1 Main Page	2
1.1 Abbreviation and Terms	2
1.2 Introduction	2
1.2.1 How to use this document	2
1.2.2 Out-of-range parameter handling	3
1.2.3 Static code analysis	3
2 Changelog	3
3 Requirement Traceability Summary	5
4 Module Documentation	23
4.1 Component Specification SIL4 Package	23
4.1.1 Detailed Description	23
4.2 RaSTA Safety and Retransmission Layer Sub-Package	24
4.2.1 Detailed Description	25
4.3 API component	25
4.3.1 Detailed Description	26
4.3.2 Function Documentation	26
4.4 Notifications component	35
4.4.1 Detailed Description	36
4.4.2 Function Documentation	36
4.5 State Machine component	39
4.5.1 Detailed Description	40
4.5.2 Function Documentation	40
4.6 Core component	59
4.6.1 Detailed Description	62
4.6.2 Function Documentation	62
4.6.3 Variable Documentation	111
4.7 Send Buffer component	111
4.7.1 Detailed Description	113
4.7.2 Function Documentation	113
4.8 Received Buffer component	132
4.8.1 Detailed Description	133
4.8.2 Function Documentation	133
4.9 Messages component	141
4.9.1 Detailed Description	143
4.9.2 Function Documentation	143
4.10 MD4 component	175
4.10.1 Detailed Description	176
4.10.2 Function Documentation	177
4.11 Diagnostics component	190
4.11.1 Detailed Description	191

4.11.2 Function Documentation	191
4.12 Adapter Interface component	202
4.12.1 Detailed Description	203
4.12.2 Function Documentation	203
4.13 Adapter Notifications component	207
4.13.1 Detailed Description	207
4.13.2 Function Documentation	208
4.14 Safety and Retransmission Layer Configuration	210
4.14.1 Detailed Description	211
4.14.2 Enumeration Type Documentation	211
4.14.3 Variable Documentation	212
4.15 Safety and Retransmission Layer API Types	212
4.15.1 Detailed Description	213
4.15.2 Enumeration Type Documentation	213
4.16 Safety and Retransmission Layer Types	214
4.16.1 Detailed Description	215
4.16.2 Enumeration Type Documentation	215
4.17 RaSTA Common Sub-Package	217
4.17.1 Detailed Description	217
4.18 Definitions component	217
4.18.1 Detailed Description	219
4.18.2 Macro Definition Documentation	219
4.18.3 Enumeration Type Documentation	220
4.19 Assert component	221
4.19.1 Detailed Description	222
4.19.2 Function Documentation	222
4.20 Helper component	231
4.20.1 Detailed Description	232
4.20.2 Function Documentation	232
4.21 Logger component	235
4.21.1 Detailed Description	235
4.21.2 Macro Definition Documentation	235
4.22 System Adapter component	236
4.22.1 Detailed Description	236
4.22.2 Function Documentation	237
5 Class Documentation	240
5.1 Md4Context Struct Reference	240
5.1.1 Detailed Description	241
5.2 radef_TransportChannelDiagnosticData Struct Reference	241
5.2.1 Detailed Description	242
5.2.2 Member Data Documentation	242

5.3 sraty_BufferUtilisation Struct Reference	243
5.3.1 Detailed Description	243
5.3.2 Member Data Documentation	243
5.4 sraty_ConnectionDiagnosticData Struct Reference	244
5.4.1 Detailed Description	245
5.4.2 Member Data Documentation	245
5.5 sraty_RedundancyChannelDiagnosticData Struct Reference	246
5.5.1 Detailed Description	247
5.5.2 Member Data Documentation	247
5.6 srcor_InputBuffer Struct Reference	248
5.6.1 Detailed Description	248
5.7 srcor_RaStaConnectionData Struct Reference	249
5.7.1 Detailed Description	250
5.7.2 Member Data Documentation	250
5.8 srcor_TemporaryBuffer Struct Reference	251
5.8.1 Detailed Description	251
5.9 srcty_ConnectionConfiguration Struct Reference	251
5.9.1 Detailed Description	252
5.9.2 Member Data Documentation	252
5.10 srcty_Md4InitValue Struct Reference	253
5.10.1 Detailed Description	253
5.11 srcty_SafetyRetransmissionConfiguration Struct Reference	253
5.11.1 Detailed Description	254
5.11.2 Member Data Documentation	254
5.12 srdia_SrConnectionDiagnostics Struct Reference	258
5.12.1 Detailed Description	258
5.13 srmd4_Md4 Struct Reference	258
5.13.1 Detailed Description	259
5.14 SrReceivedBuffer Struct Reference	259
5.14.1 Detailed Description	259
5.15 SrSendBuffer Struct Reference	260
5.15.1 Detailed Description	260
5.16 SrSendMessage Struct Reference	261
5.16.1 Detailed Description	261
5.17 srtyp_ProtocolVersion Struct Reference	261
5.17.1 Detailed Description	261
5.17.2 Member Data Documentation	262
5.18 srtyp_SrMessage Struct Reference	262
5.18.1 Detailed Description	262
5.19 srtyp_SrMessageHeader Struct Reference	262
5.19.1 Detailed Description	263
5.20 srtyp_SrMessageHeaderCreate Struct Reference	263

5.20.1 Detailed Description	264
5.21 srtyp_SrMessageHeaderUpdate Struct Reference	264
5.21.1 Detailed Description	264
5.22 srtyp_SrMessagePayload Struct Reference	264
5.22.1 Detailed Description	265
6 File Documentation	265
6.1 raas_rasta_assert.h File Reference	265
6.1.1 Detailed Description	266
6.2 radef_rasta_definitions.h File Reference	266
6.2.1 Detailed Description	268
6.3 rahlp_rasta_helper.h File Reference	269
6.3.1 Detailed Description	269
6.4 ralog_rasta_logger.h File Reference	270
6.4.1 Detailed Description	270
6.5 rasys_rasta_system_adapter.h File Reference	271
6.5.1 Detailed Description	272
6.6 raas_rasta_assert.c File Reference	272
6.6.1 Detailed Description	273
6.7 rahlp_rasta_helper.c File Reference	273
6.7.1 Detailed Description	274
6.8 ralog_rasta_logger.c File Reference	274
6.8.1 Detailed Description	275
6.9 sradin_sr_adapter_interface.h File Reference	275
6.9.1 Detailed Description	276
6.10 sradno_sr_adapter_notifications.h File Reference	277
6.10.1 Detailed Description	278
6.11 srapi_sr_api.h File Reference	278
6.11.1 Detailed Description	279
6.12 sraty_sr_api_types.h File Reference	280
6.12.1 Detailed Description	281
6.13 srcty_sr_config_types.h File Reference	281
6.13.1 Detailed Description	283
6.14 srnot_sr_notifications.h File Reference	283
6.14.1 Detailed Description	284
6.15 sradno_sr_adapter_notifications.c File Reference	285
6.15.1 Detailed Description	286
6.16 srapi_sr_api.c File Reference	286
6.16.1 Detailed Description	287
6.17 srcor_sr_core.c File Reference	288
6.17.1 Detailed Description	291
6.18 srcor_sr_core.h File Reference	291

6.18.1 Detailed Description	293
6.19 srcty_sr_config_types.c File Reference	294
6.19.1 Detailed Description	295
6.20 srdia_sr_diagnostics.c File Reference	295
6.20.1 Detailed Description	297
6.21 srdia_sr_diagnostics.h File Reference	297
6.21.1 Detailed Description	299
6.22 srmd4_sr_md4.c File Reference	299
6.22.1 Detailed Description	301
6.22.2 Enumeration Type Documentation	301
6.23 srmd4_sr_md4.h File Reference	302
6.23.1 Detailed Description	303
6.24 srmsg_sr_messages.c File Reference	303
6.24.1 Detailed Description	306
6.24.2 Variable Documentation	307
6.25 srmsg_sr_messages.h File Reference	307
6.25.1 Detailed Description	309
6.26 srrece_sr_received_buffer.c File Reference	309
6.26.1 Detailed Description	310
6.27 srrece_sr_received_buffer.h File Reference	311
6.27.1 Detailed Description	312
6.28 srsend_sr_send_buffer.c File Reference	312
6.28.1 Detailed Description	314
6.28.2 Variable Documentation	314
6.29 srsend_sr_send_buffer.h File Reference	315
6.29.1 Detailed Description	316
6.30 srstm_sr_state_machine.c File Reference	316
6.30.1 Detailed Description	318
6.31 srstm_sr_state_machine.h File Reference	318
6.31.1 Detailed Description	319
6.32 srtyp_sr_types.h File Reference	319
6.32.1 Detailed Description	321
6.33 srcfg_sr_config.h File Reference	321
6.33.1 Detailed Description	322
6.34 srcfg_sr_config.c File Reference	322
6.34.1 Detailed Description	323
Index	325

1 Main Page

1.1 Abbreviation and Terms

This document uses common terms and abbreviations that are defined as follows:

- File: Describes a component as defined in the *Software Architecture and Design Specification*
- Module: Synonym for component
- Member: Function or enumeration which is part of a component
- Class: Describes a structure (struct)

These terms are defined by doxygen and cannot be adjusted.

1.2 Introduction

This document aims to specify the software design of the individual components of the RaSTA protocol stack. The [Component Specification SIL4 Package](#) contains all component specifications of the following safety related layers, developed according to SIL4 requirements:

- RaSTA Safety and Retransmission Layer
- RaSTA Common Package

As the specification of the components is directly embedded in the source code, this documentation is automatically created with the use of Doxygen.

1.2.1 How to use this document

Chapter [Changelog](#) shows the change history of all files. The change management starting from initial version V1.0 requires all code changes to be marked in the file header including the following information: ID of the change issue, brief description of the changes, date and author.

Chapter [Requirement Traceability Summary](#) shows a list of all software requirements implemented by a component member. This can be a function, variable or definition for example. All linked requirements can be found in the *SBB-RaSTA-018 - Software Architecture and Design Specification* as well as the *SBB-RaSTA-020 - Software Interface Specification*.

Chapter Module Documentation and its subchapter [Component Specification SIL4 Package](#) presents the detailed specification of the components.

The Class Documentation as well as the File Documentation show further information about the components. This can help to find some additional information, but is not relevant for the design specification.

1.2.2 Out-of-range parameter handling

For input parameter checks, a `radef_kInvalidParameter` fatal error is raised if a parameter is invalid. Reasons for invalid parameters are:

- value out of range
- null-pointer

This definition is valid for all parameters where nothing else is specified.

All enumerations contains an additional min & max entry for range checks. This entries are never used expect for range checks.

1.2.3 Static code analysis

For static code analysis different tools are used. For all this tools, it is possible to suppress errors. This is done using the following commands:

- pc-lint
 - `//lint -esym(error code, symbol) (description)` --> ex. `//lint -esym(788, radef_kMax)` (used only for parameter range checking)
 - `//lint -save -e(error code) (description)` --> ex. `//lint -save -e9045` (structures are defined globally)
 - `//lint -restore` --> ex. to finish a saved suppression
- cpp lint
 - `// NOLINT(error)` --> ex. `// NOLINT(build/include_subdir)`
- cpp check
 - `// cppcheck-suppress error` --> ex. `// cppcheck-suppress variableScope`

2 Changelog

File `raas_rasta_assert.c`

-: Initial version (-, -)

File `raas_rasta_assert.h`

-: Initial version (-, -)

File `radef_rasta_definitions.h`

-: Initial version (-, -)

File `rahlp_rasta_helper.c`

-: Initial version (-, -)

File `rahlp_rasta_helper.h`

-: Initial version (-, -)

File `ralog_rasta_logger.c`

-: Initial version (-, -)

File ralog_rasta_logger.h

-: Initial version (-, -)

File rasys_rasta_system_adapter.h

-: Initial version (-, -)

File sradin_sr_adapter_interface.h

-: Initial version (-, -)

File sradno_sr_adapter_notifications.c

-: Initial version (-, -)

SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4932: sr_adapter_notification module functions are not error tolerant (08.12.2022, N. Andres)

File sradno_sr_adapter_notifications.h

-: Initial version (-, -)

SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4932: sr_adapter_notification module functions are not error tolerant (08.12.2022, N. Andres)

File srapi_sr_api.c

-: Initial version (-, -)

File srapi_sr_api.h

-: Initial version (-, -)

File sraty_sr_api_types.h

-: Initial version (-, -)

File srcfg_sr_config.c

-: Initial version (-, -)

File srcfg_sr_config.h

-: Initial version (-, -)

File srcor_sr_core.c

-: Initial version (-, -)

SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4933: Duplicated sent of diagnostic notification in sr_core when closing the connection (06.12.2022, N. Andres)

File srcor_sr_core.h

-: Initial version (-, -)

SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4933: Duplicated sent of diagnostic notification in sr_core when closing the connection (06.12.2022, N. Andres)

File srcty_sr_config_types.c

-: Initial version (-, -)

File srcty_sr_config_types.h

-: Initial version (-, -)

File srdia_sr_diagnostics.c

-: Initial version (-, -)

File srdia_sr_diagnostics.h

-: Initial version (-, -)

File srmd4_sr_md4.c

-: Initial version (-, -)

File srmd4_sr_md4.h

-: Initial version (-, -)

File srmsg_sr_messages.c

-: Initial version (-, -)

SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4935: Messages module must check message size and payload data size correctly (07.12.2022, N. Andres)

File srmsg_sr_messages.h

-: Initial version (-, -)

SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4935: Messages module must check message size and payload data size correctly (07.12.2022, N. Andres)

File srnot_sr_notifications.h

-: Initial version (-, -)

File srrece_sr_received_buffer.c

-: Initial version (-, -)

File srrece_sr_received_buffer.h

-: Initial version (-, -)

File srsend_sr_send_buffer.c

-: Initial version (-, -)

File srsend_sr_send_buffer.h

-: Initial version (-, -)

File srstm_sr_state_machine.c

-: Initial version (-, -)

File srstm_sr_state_machine.h

-: Initial version (-, -)

File srtyp_sr_types.h

-: Initial version (-, -)

3 Requirement Traceability Summary

Member AddTimeToTimingDistribution (uint32_t *const distribution_array, const uint32_t time_value)

RASW-645 Update Connection Diagnostics Function

RASW-433 Diagnostic Timing Interval

Member AddToBuffer (const uint32_t connection_id, const srtyp_SrMessage *const message)

RASW-596 Add to Buffer Function

Member CalculateTimeliness (const uint32_t connection_id, const srtyp_SrMessageHeader *const msg_hdr, const uint32_t current_time)

RASW-580 Receive Message Function

RASW-806 Timeliness Check

RASW-808 Timer Ti

Member CheckConfirmedSequenceNumber (const uint32_t connection_id, const srtyp_SrMessageHeader *const msg_hdr)

RASW-580 Receive Message Function

RASW-804 Sequence Integrity of the Confirmed Sequence Number

Member CheckConfirmedTimeStamp (const uint32_t connection_id, const srtyp_SrMessageHeader *const msg_hdr)

RASW-580 Receive Message Function

RASW-822 Sequence Integrity of the Confirmed Time Stamp

Member `CheckConnectionConfigurations` (const uint32_t number_of_connections, const srcty_ConnectionConfiguration *const connection_configurations)
RASW-573 Is Configuration Valid Function

Member `CheckSequenceNumber` (const uint32_t connection_id, const srtyp_SrMessageHeader *const msg_hdr)
RASW-580 Receive Message Function
RASW-805 Sequence Number Check

Member `CheckSequenceNumberRange` (const uint32_t connection_id, const srtyp_SrMessageHeader *const msg_hdr)
RASW-580 Receive Message Function
RASW-803 Sequence Number Range Check

Member `CheckTimeStamp` (const uint32_t connection_id, const srtyp_SrMessageHeader *const msg_hdr)
RASW-580 Receive Message Function
RASW-806 Timeliness Check

Member `ClearMd4ContextData` (Md4Context *const md4_context)
RASW-633 Calculate MD4 Function

Member `CloseConnection` (const uint32_t connection_id, const sraty_DiscReason disconnect_reason, const bool is_incoming_message)
RASW-563 Process Connection State Machine Function

Member `CloseRedundancyChannel` (const uint32_t connection_id, const bool is_incoming_message)
RASW-563 Process Connection State Machine Function

Module `common_assert`
RASW-533 Component rasta_assert Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level

Module `common_definitions`
RASW-525 Component rasta_definitions Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level

Module `common_helper`
RASW-818 Component rasta_Helper Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level

Module `common_logger`
RASW-540 Component rasta_logger Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level

Module `common_system_adapter`
RASW-527 Component rasta_system_adapter Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level

Member `CopyTempBufferIntoConnectionBuffer` (const uint32_t connection_id)
RASW-603 Prepare Buffer for Retransmission Function

Member `CopyToContextBuffer` (const uint8_t *const source, const uint32_t size, const uint32_t md4_context_buffer_start_index, Md4Context *const md4_context)
RASW-633 Calculate MD4 Function

Member `CreateNewDataMsgAndAddToTempBuffer` (const srtyp_SrMessage *const sr_message, srtyp_SrMessageHeaderCreate *const new_msg_header)
RASW-603 Prepare Buffer for Retransmission Function

Member `CreateNewHbMsgAndAddToTempBuffer` (srtyp_SrMessageHeaderCreate *const new_msg_header)
RASW-603 Prepare Buffer for Retransmission Function

Member `CreateNewRetrDataMsgAndAddToTempBuffer` (const `srtyp_SrMessage` *const `sr_message`, `srtyp_SrMessageHeaderCreate` *const `new_msg_header`)
RASW-603 Prepare Buffer for Retransmission Function

Member `CreateNewRetrReqMsgAndAddToTempBuffer` (`srtyp_SrMessageHeaderCreate` *const `new_msg_header`)
RASW-603 Prepare Buffer for Retransmission Function

Member `FunctionF` (const `uint32_t` `x`, const `uint32_t` `y`, const `uint32_t` `z`)
RASW-633 Calculate MD4 Function

Member `FunctionG` (const `uint32_t` `x`, const `uint32_t` `y`, const `uint32_t` `z`)
RASW-633 Calculate MD4 Function

Member `FunctionH` (const `uint32_t` `x`, const `uint32_t` `y`, const `uint32_t` `z`)
RASW-633 Calculate MD4 Function

Member `GeneralMessageCheck` (const `uint32_t` `connection_id`, const `srtyp_SrMessage` *const `msg`, `srtyp_SrMessageHeader` *const `msg_hdr`)
RASW-580 Receive Message Function

Member `Get` (const `uint8_t` `index_32_bit`, const `Md4Context` *const `ctx`)
RASW-633 Calculate MD4 Function

Member `GetCurrentSequenceNumberAndIncrementNumber` (const `uint32_t` `connection_id`)
RASW-584 Send Data Message Function
RASW-582 Send ConnReq Message Function
RASW-583 Send ConnResp Message Function
RASW-585 Send DiscReq Message Function
RASW-586 Send Heartbeat Message Function
RASW-588 Send RetrReq Message Function
RASW-570 Handle Retransmission Request Function

Member `GetSafetyCodeLength` (`void`)
RASW-168 Safety Code

Member `GetUint16FromMessage` (const `srtyp_SrMessage` *const `sr_message`, const `uint16_t` `position`)
RASW-157 Endian Definition

Member `GetUint32FromMessage` (const `srtyp_SrMessage` *const `sr_message`, const `uint16_t` `position`)
RASW-157 Endian Definition

Member `IncrementReceivedBufferIndexAndHandleOverflow` (`uint16_t` *const `bufferIndex`, const `uint16_t` `increment`)
RASW-608 Add to Buffer Function
RASW-613 Read from Buffer Function

Member `IncrementSendBufferIndexAndHandleOverflow` (`uint16_t` *const `bufferIndex`, const `uint16_t` `increment`)
RASW-604 Read Message to Send Function
RASW-603 Prepare Buffer for Retransmission Function
RASW-605 Remove from Buffer Function

Member `InitBuffer` (const `uint32_t` `connection_id`)
RASW-601 Init Buffer Function

Member `IsMessageTimeoutRelated` (const `srtyp_SrMessageType` `message_type`)
RASW-579 Process Received Messages Function

Member `IsMessageTypeValid` (`srtyp_SrMessageType` `message_type`)
RASW-616 Check Message Function

Member **Md4Body** (**Md4Context** *const ctx, const uint8_t *const data, const uint32_t number_of_blocks)

RASW-633 Calculate MD4 Function

RASW-634 Safety Code

Member **Md4Final** (**Md4Context** *ctx, uint8_t *result)

RASW-633 Calculate MD4 Function

RASW-634 Safety Code

Member **Md4Update** (**Md4Context** *const ctx, const uint8_t *const data, const uint32_t size)

RASW-633 Calculate MD4 Function

RASW-634 Safety Code

Member **ProcessStateClosedEvents** (const uint32_t connection_id, const srtyp_ConnectionEvents event)

RASW-563 Process Connection State Machine Function

RASW-560 sr_state_machine Events

Member **ProcessStateDownEvents** (const uint32_t connection_id, const srtyp_ConnectionEvents event)

RASW-563 Process Connection State Machine Function

RASW-560 sr_state_machine Events

RASW-755 Process State Machine Receive ConnReq Message Sequence

Member **ProcessStateRetransRequestEvents** (const uint32_t connection_id, const srtyp_ConnectionEvents event, const bool sequence_number_in_seq)

RASW-563 Process Connection State Machine Function

RASW-560 sr_state_machine Events

Member **ProcessStateRetransRunningEvents** (const uint32_t connection_id, const srtyp_ConnectionEvents event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)

RASW-563 Process Connection State Machine Function

RASW-560 sr_state_machine Events

Member **ProcessStateStartEvents** (const uint32_t connection_id, const srtyp_ConnectionEvents event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)

RASW-563 Process Connection State Machine Function

RASW-560 sr_state_machine Events

RASW-757 Process State Machine Receive ConnResp Message Sequence

Member **ProcessStateUpEvents** (const uint32_t connection_id, const srtyp_ConnectionEvents event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)

RASW-563 Process Connection State Machine Function

RASW-560 sr_state_machine Events

RASW-759 Process State Machine Receive Data Message (in Sequence) Sequence

RASW-761 Process State Machine Receive RetrReq Message (in Sequence) Sequence

RASW-763 Process State Machine Receive RetrReq Message (not in Sequence) Sequence

Member **raas_AssertNotNull** (const void *const pointer, const radef_RaStaReturnCode error_reason)

RASW-534 Assert not Null Function

RASW-521 Input Parameter Check

Member **raas_AssertTrue** (const bool condition, const radef_RaStaReturnCode error_reason)

RASW-535 Assert True Function

RASW-521 Input Parameter Check

Member **raas_AssertU16InRange** (const uint16_t value, const uint16_t min_value, const uint16_t max_value, const radef_RaStaReturnCode error_reason)

RASW-536 Assert U16 in Range Function

RASW-521 Input Parameter Check

Member `raas_AssertU32InRange` (const uint32_t value, const uint32_t min_value, const uint32_t max_value, const radef_RaStaReturnCode error_reason)

RASW-537 Assert U32 in Range Function
RASW-521 Input Parameter Check

Member `raas_AssertU8InRange` (const uint8_t value, const uint8_t min_value, const uint8_t max_value, const radef_RaStaReturnCode error_reason)

RASW-538 Assert U8 in Range Function
RASW-521 Input Parameter Check

Member `radef_RaStaReturnCode`

RASW-483 Enum RaSta Return Code Structure
RASW-503 Enum RaSta Return Code Usage

Class `radef_TransportChannelDiagnosticData`

RASW-474 Struct Transport Channel Diagnostic Data Structure

Member `radef_TransportChannelDiagnosticData::n_diagnosis`

RASW-469 N diagnosis

Member `radef_TransportChannelDiagnosticData::n_missed`

RASW-473 N missed

Member `radef_TransportChannelDiagnosticData::t_drift`

RASW-472 T drift

Member `radef_TransportChannelDiagnosticData::t_drift2`

RASW-467 T drift

Member `rahlp_IsU16InRange` (const uint16_t value, const uint16_t min_value, const uint16_t max_value)

RASW-821 Is U16 in Range Function
RASW-521 Input Parameter Check

Member `rahlp_IsU32InRange` (const uint32_t value, const uint32_t min_value, const uint32_t max_value)

RASW-820 Is U32 in Range Function
RASW-521 Input Parameter Check

Member `rasys_FatalError` (const radef_RaStaReturnCode error_reason)

RASW-528 Fatal Error Function
RASW-417 Fatal Error Handling Function Structure
RASW-416 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-520 Error Handling

Member `rasys_GetRandomNumber` (void)

RASW-529 Get Random Number Function
RASW-414 Get Random Number Function Structure
RASW-413 Random Number

Member `rasys_GetTimerGranularity` (void)

RASW-530 Get Timer Granularity Function
RASW-420 Get Timer Granularity Function Structure
RASW-419 Timer Granularity

Member `rasys_GetTimerValue` (void)

RASW-531 Get Timer Value Function
RASW-410 Get Timer Value Function Structure
RASW-422 Timer Value

Member `ReceivedFlowControlCheck` (const uint32_t connection_id, const srtyp_SrMessageType message_type)
RASW-579 Process Received Messages Function

Member `SendPendingMessagesWithFlowControlAllowed` (const uint32_t connection_id)
RASW-587 Send Pending Messages Function

Member `Set` (const uint8_t index_32_bit, const uint8_t *const data, Md4Context *const ctx)
RASW-633 Calculate MD4 Function

Member `SetConnectionEvent` (const srtyp_SrMessageType message_type, srtyp_ConnectionEvents *const connection_event)
RASW-580 Receive Message Function

Member `SetContextBuffer` (const uint8_t value, const uint32_t size, const uint32_t md4_context_buffer_start_index, Md4Context *const md4_context)
RASW-633 Calculate MD4 Function

Member `SetMessageHeaderInMessage` (const uint16_t message_length, const uint16_t message_type, const srtyp_SrMessageHeaderCreate message_header, srtyp_SrMessage *const sr_message)
RASW-157 Endian Definition
RASW-160 General PDU Message Structure
RASW-161 Message Type
RASW-162 Receiver Identification
RASW-163 Sender Identification
RASW-167 Confirmed Time Stamp

Member `SetPayloadDataInMessage` (const uint16_t position, const srtyp_SrMessagePayload *const message_payload, srtyp_SrMessage *const sr_message)
RASW-157 Endian Definition
RASW-160 General PDU Message Structure

Member `SetProtocolVersionInMessage` (const srtyp_ProtocolVersion protocol_version, srtyp_SrMessage *const sr_message)
RASW-157 Endian Definition
RASW-170 Connection Request Message Structure
RASW-171 Connection Response Message Structure
RASW-173 Protocol Version

Member `SetUint16InMessage` (const uint16_t position, const uint16_t data, srtyp_SrMessage *const sr_message)
RASW-157 Endian Definition

Member `SetUint32InMessage` (const uint16_t position, const uint32_t data, srtyp_SrMessage *const sr_message)
RASW-157 Endian Definition

Member `SetUint64InMessage` (const uint16_t position, const uint64_t data, srtyp_SrMessage *const sr_message)
RASW-157 Endian Definition

Module `sr_adapter_interface`
RASW-647 Component sr_adapter_interface Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level

Module `sr_adapter_notifications`
RASW-654 Component sr_adapter_notifications Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_api

RASW-543 Component sr_api Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_api_types

RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-491 Enum Connection States Structure
RASW-489 Enum Disc Reason Structure
RASW-461 Struct Buffer Utilisation Structure
RASW-470 Struct Connection Diagnostic Data Structure
RASW-475 Struct Redundancy Channel Diagnostic Data Structure

Module sr_config

RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-487 Enum Safety Code Type Structure
RASW-423 Struct Connection Configuration Structure
RASW-437 Struct MD4 Initial Value Structure
RASW-427 Struct SafetyRetransmissionConfiguration Structure

Module sr_core

RASW-565 Component sr_core Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_diagnostics

RASW-636 Component sr_diagnostics Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_md4

RASW-632 Component sr_md4 Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_messages

RASW-615 Component sr_messages Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_notifications

RASW-553 Component sr_notifications Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level

Module sr_receiveBuf

RASW-607 Component sr_received_buffer Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_sendBuf

RASW-595 Component sr_send_buffer Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_sm

RASW-559 Component sr_state_machine Overview
RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-520 Error Handling
RASW-521 Input Parameter Check

Module sr_types

RASW-518 Safety and Retransmission Layer Safety Integrity Level
RASW-560 sr_state_machine Events

Member sradin_CloseRedundancyChannel (const uint32_t redundancy_channel_id)

RASW-650 Close Redundancy Channel Function
RASW-368 Close Redundancy Channel Function Structure
RASW-367 Redundancy Channel Id

Member sradin_Init (void)

RASW-648 Init sr_adapter_interface Function
RASW-353 Initialization Function Structure

Member sradin_OpenRedundancyChannel (const uint32_t redundancy_channel_id)

RASW-649 Open Redundancy Channel Function
RASW-369 Open Redundancy Channel Function Structure
RASW-367 Redundancy Channel Id

Member sradin_ReadMessage (const uint32_t redundancy_channel_id, const uint16_t buffer_size, uint16_t *const message_size, uint8_t *const message_buffer)

RASW-652 Read Message Function
RASW-375 Read Message Function Structure
RASW-381 Redundancy Channel Id
RASW-379 Buffer Size
RASW-372 Message Size
RASW-371 Message Buffer
RASW-374 Error Code
RASW-503 Enum RaSta Return Code Usage

Member sradin_SendMessage (const uint32_t redundancy_channel_id, const uint16_t message_size, const uint8_t *const message_data)

RASW-651 Send Message Function
RASW-364 Send Message Function Structure
RASW-363 Redundancy Channel Id
RASW-387 Message Size
RASW-385 Message Data

Member sradno_DiagnosticNotification (const uint32_t red_channel_id, const uint32_t tr_channel_id, const radef_TransportChannelDiagnosticData tr_channel_diagnostic_data)

RASW-656 Diagnostic Notification
RASW-332 Diagnostic Notification Structure

RASW-331 Redundancy Channel Id
RASW-334 Transport Channel Id
RASW-333 Transport Channel Diagnostic Data
RASW-901 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-751 Redundancy Diagnostic Notification Sequence

Member `sradno_MessageReceivedNotification (const uint32_t red_channel_id)`

RASW-655 Message Received Notification
RASW-335 Message Received Notification Structure
RASW-338 Redundancy Channel Id
RASW-900 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-747 Message Received Notification Sequence
RASW-769 Received Message Polling Sequence

Member `srapi_CheckTimings (void)`

RASW-551 Check Timings Function
RASW-319 Check Timings Function Structure
RASW-317 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-753 Check Timings Sequence
RASW-769 Received Message Polling Sequence
RASW-773 Check Timings Message Timeout / Heartbeat Sequence

Member `srapi_CloseConnection (const uint32_t connection_id, const uint16_t detailed_reason)`

RASW-547 Close Connection Function
RASW-318 Close Connection Function Structure
RASW-316 Connection Identification
RASW-309 Detailed Reason
RASW-308 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-739 Close Connection Sequence

Member `srapi_GetConnectionState (const uint32_t connection_id, sraty_ConnectionStates *const connection_state, sraty_BufferUtilisation *const buffer_utilisation, uint16_t *const opposite_buffer_size)`

RASW-550 Get Connection State Function
RASW-288 Get Connection State Function Structure
RASW-287 Connection Identification
RASW-282 Connection State
RASW-281 Buffer Utilisation
RASW-284 Opposite Buffer Size
RASW-283 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-745 Get Connection State Sequence

Member `sraPI_GetInitializationState` (`void`)

RASW-545 Get Initialization State Function
RASW-306 Get Initialization State Function Structure
RASW-305 Error Code
RASW-503 Enum RaSta Return Code Usage

Member `sraPI_Init` (`const srcty_SafetyRetransmissionConfiguration *const safety_retransmission_configuration`)

RASW-544 Init sr_api Function
RASW-267 Initialization Function Structure
RASW-292 Configuration SafRetL
RASW-290 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-735 Init SafRetL Sequence

Member `sraPI_OpenConnection` (`const uint32_t sender_id, const uint32_t receiver_id, const uint32_t network_id, uint32_t *const connection_id`)

RASW-546 Open Connection Function
RASW-303 Open Connection Function Structure
RASW-301 Sender Identification
RASW-300 Receiver Identification
RASW-324 Network Identification
RASW-322 Connection Identification
RASW-314 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-737 Open Connection Sequence

Member `sraPI_ReadData` (`const uint32_t connection_id, const uint16_t buffer_size, uint16_t *const message_size, uint8_t *const message_buffer`)

RASW-549 Read Data Function
RASW-274 Read Data Function Structure
RASW-273 Connection Identification
RASW-269 Buffer Size
RASW-268 Message Size
RASW-270 Message Buffer
RASW-286 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-743 Read Data Sequence

Member `sraPI_SendData` (`const uint32_t connection_id, const uint16_t message_size, const uint8_t *const message_data`)

RASW-548 Send Data Function
RASW-310 Send Data Function Structure
RASW-275 Connection Identification
RASW-277 Message Size
RASW-276 Message Data
RASW-272 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-741 Send Data Sequence

Class `srtty_BufferUtilisation`

RASW-461 Struct Buffer Utilisation Structure

Member `srtty_BufferUtilisation::receive_buffer_free`

RASW-463 Free Receive Buffer Entries

Member `srtty_BufferUtilisation::receive_buffer_used`

RASW-464 Used Receive Buffer Entries

Member `srtty_BufferUtilisation::send_buffer_free`

RASW-465 Free Send Buffer Entries

Member `srtty_BufferUtilisation::send_buffer_used`

RASW-460 Used Send Buffer Entries

Class `srtty_ConnectionDiagnosticData`

RASW-470 Struct Connection Diagnostic Data Structure

Member `srtty_ConnectionDiagnosticData::ec_address`

RASW-482 EC address

Member `srtty_ConnectionDiagnosticData::ec_csn`

RASW-479 EC CSN

Member `srtty_ConnectionDiagnosticData::ec_safety`

RASW-468 EC safety

Member `srtty_ConnectionDiagnosticData::ec_sn`

RASW-480 EC SN

Member `srtty_ConnectionDiagnosticData::ec_type`

RASW-481 EC type

Member `srtty_ConnectionDiagnosticData::t_alive_distribution` [RADEF_DIAGNOSTIC_TIMING_←
DISTRIBUTION_INTERVALS]

RASW-477 Alive Time Distribution

Member `srtty_ConnectionDiagnosticData::t_rtd_distribution` [RADEF_DIAGNOSTIC_TIMING_←
DISTRIBUTION_INTERVALS]

RASW-478 Round Trip Delay Time Distribution

Member `srtty_ConnectionStates`

RASW-491 Enum Connection States Structure

Member `srtty_DiscReason`

RASW-489 Enum Disc Reason Structure

Class `srtty_RedundancyChannelDiagnosticData`

RASW-475 Struct Redundancy Channel Diagnostic Data Structure

Member `srtty_RedundancyChannelDiagnosticData::n_diagnosis`

RASW-469 N diagnosis

Member `srtty_RedundancyChannelDiagnosticData::n_missed`

RASW-473 N missed

Member `srtty_RedundancyChannelDiagnosticData::t_drift`

RASW-472 T drift

Member `srtty_RedundancyChannelDiagnosticData::t_drift2`

RASW-467 T drift2

Member `srtty_RedundancyChannelDiagnosticData::transport_channel_id`

RASW-471 Transport Channel Id

Member `srcor_ClearInputBufferMessagePendingFlag (const uint32_t connection_id)`

RASW-566 Clear Input Buffer Message Pending Flag Function

Member `srcor_CloseRedundancyChannel (const uint32_t connection_id)`

RASW-826 Close Redundancy Channel Function

Member `srcor_GetBufferSizeAndUtilisation (const uint32_t connection_id, sraty_BufferUtilisation *const buffer_utilisation, uint16_t *const opposite_buffer_size)`

RASW-567 Get Buffer Size and Utilisation Function

Member `srcor_GetConnectionId (const uint32_t sender_id, const uint32_t receiver_id, uint32_t *const connection_id)`

RASW-568 Get Connection ID Function

Member `srcor_GetReceivedMessagePendingFlag (const uint32_t connection_id)`

RASW-569 Get Received Message Pending Flag Function

Member `srcor_HandleRetrReq (const uint32_t connection_id)`

RASW-570 Handle Retransmission Request Function

RASW-777 Handle Retransmission Request Sequence

Member `srcor_Init (const srcty_SafetyRetransmissionConfiguration *const sr_layer_configuration)`

RASW-571 Init sr_core Function

Member `srcor_InitRaStaConnData (const uint32_t connection_id)`

RASW-572 Init RaSTA Connection Data Function

Member `srcor_IsConfigurationValid (const srcty_SafetyRetransmissionConfiguration *const sr_layer_configuration)`

RASW-573 Is Configuration Valid Function

Member `srcor_IsConnRoleServer (const uint32_t connection_id)`

RASW-574 Is Connection Role Server Function

Member `srcor_IsHeartbeatInterval (const uint32_t connection_id)`

RASW-575 Is Heartbeat Interval Function

RASW-807 Timer Th

Member `srcor_IsMessageTimeout (const uint32_t connection_id)`

RASW-576 Is Message Timeout Function

RASW-808 Timer Ti

Member `srcor_IsProtocolVersionAccepted (const uint32_t connection_id)`

RASW-577 Is Protocol Version Accepted Function

Member `srcor_IsReceivedMsgPendingAndBuffersNotFull (const uint32_t connection_id)`

RASW-830 Is Received Message Pending And Buffers Not Full Function

Member `srcor_IsRetrReqSequenceNumberAvailable (const uint32_t connection_id)`

RASW-578 Is Retransmission Request Sequence Number Available Function

Member `srcor_ProcessReceivedMessage (const uint32_t connection_id)`

RASW-579 Process Received Messages Function

RASW-775 Process Received Message Sequence

Member `srcor_ReceiveMessage (const uint32_t connection_id, srtyp_ConnectionEvents *const connection_event, bool *const sequence_number_in_seq, bool *const confirmed_time_stamp_in_seq)`

RASW-580 Receive Message Function

RASW-771 Receive Message Sequence

Member `srcor_SendConnectionStateNotification (const uint32_t connection_id, const sraty_ConnectionStates connection_state, const sraty_DiscReason disconnect_reason)`

RASW-581 Send Connection State Notification Function

Member `srcor_SendConnReqMessage (const uint32_t connection_id)`

RASW-582 Send ConnReq Message Function

Member `srcor_SendConnRespMessage (const uint32_t connection_id)`

RASW-583 Send ConnResp Message Function

Member `srcor_SendDataMessage (const uint32_t connection_id)`

RASW-584 Send Data Message Function

Member `srcor_SendDiscReqMessage (const uint32_t connection_id, const sraty_DiscReason disconnect_reason)`

RASW-585 Send DiscReq Message Function

RASW-767 Send Disconnection Request Message Sequence

Member `srcor_SendHbMessage (const uint32_t connection_id)`

RASW-586 Send Heartbeat Message Function

Member `srcor_SendPendingMessages (const uint32_t connection_id)`

RASW-587 Send Pending Messages Function

RASW-765 Send Pending Messages Sequence

Member `srcor_SendRetrReqMessage (const uint32_t connection_id)`

RASW-588 Send RetrReq Message Function

Member `srcor_SetDiscDetailedReason (const uint32_t connection_id, const uint16_t detailed_disconnect_reason)`

RASW-589 Set Disconnection Detailed Reason Function

Member `srcor_SetReceivedMessagePendingFlag (const uint32_t connection_id)`

RASW-590 Set Received Message Pending Flag Function

Member `srcor_UpdateConfirmedRxSequenceNumber (const uint32_t connection_id)`

RASW-591 Update Confirmed Rx Sequence Number Function

Member `srcor_UpdateConfirmedTxSequenceNumber (const uint32_t connection_id)`

RASW-592 Update Confirmed Tx Sequence Number Function

Member `srcor_WriteMessagePayloadToTemporaryBuffer (const uint32_t connection_id, const uint16_t message_payload_size, const uint8_t *const message_payload)`

RASW-593 Write Message Payload to Temporary Buffer Function

Class `srcty_ConnectionConfiguration`

RASW-423 Struct Connection Configuration Structure

Member `srcty_ConnectionConfiguration::connection_id`

RASW-426 Connection Id

Member `srcty_ConnectionConfiguration::receiver_id`

RASW-435 Receiver Id

Member `srcty_ConnectionConfiguration::sender_id`

RASW-425 Sender Id

Class `srcty_Md4InitValue`

RASW-437 Struct MD4 Initial Value Structure

RASW-432 Init Value A, B, C, D

Member `srcty_SafetyCodeType`

RASW-487 Enum Safety Code Type Structure

Class `srcty_SafetyRetransmissionConfiguration`

RASW-427 Struct SafetyRetransmissionConfiguration Structure

Member `srcty_SafetyRetransmissionConfiguration::connection_configurations` [RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS]
RASW-431 Connection Configurations

Member `srcty_SafetyRetransmissionConfiguration::diag_timing_distr_intervals` [RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_ARRAY_SIZE]
RASW-433 Diagnostic Timing Interval

Member `srcty_SafetyRetransmissionConfiguration::m_w_a`
RASW-442 MWA

Member `srcty_SafetyRetransmissionConfiguration::md4_initial_value`
RASW-428 MD4 Initial Value

Member `srcty_SafetyRetransmissionConfiguration::n_diag_window`
RASW-438 N diagWindow

Member `srcty_SafetyRetransmissionConfiguration::n_max_packet`
RASW-440 N maxPacket

Member `srcty_SafetyRetransmissionConfiguration::n_send_max`
RASW-441 N sendmax

Member `srcty_SafetyRetransmissionConfiguration::number_of_connections`
RASW-436 Number of Connections

Member `srcty_SafetyRetransmissionConfiguration::rasta_network_id`
RASW-446 RaSTA Network Id

Member `srcty_SafetyRetransmissionConfiguration::safety_code_type`
RASW-443 Safety Code Type

Member `srcty_SafetyRetransmissionConfiguration::t_h`
RASW-444 T h

Member `srcty_SafetyRetransmissionConfiguration::t_max`
RASW-445 T max

Member `srdia_AreDiagnosticTimingIntervalsValid` (const uint32_t t_max, const uint32_t *const diag_timing_distr_intervals)
RASW-817 Are Diagnostic Timing Intervals Valid Function

Member `srdia_IncAddressErrorCounter` (const uint32_t connection_id)
RASW-637 Inc Address Error Counter Function

Member `srdia_IncConfirmedSequenceNumberErrorCounter` (const uint32_t connection_id)
RASW-638 Inc Confirmed Sequence Number Error Counter Function

Member `srdia_IncSafetyCodeErrorCounter` (const uint32_t connection_id)
RASW-639 Inc Safety Code Error Counter Function

Member `srdia_IncSequenceNumberErrorCounter` (const uint32_t connection_id)
RASW-640 Inc Sequence Number Error Counter Function

Member `srdia_IncTypeErrorHandler` (const uint32_t connection_id)
RASW-641 Inc Type Error Counter Function

Member `srdia_Init` (const uint32_t configured_connections, const uint32_t t_max, const uint32_t n_diag_window, const uint32_t *const diag_timing_distr_intervals)
RASW-642 Init sr_diagnostic Function

Member `srdia_InitConnectionDiagnostics` (const uint32_t connection_id)
RASW-643 Init Connection Diagnostics Function

Member `srdia_SendDiagnosticNotification` (const uint32_t connection_id)
RASW-644 Send Diagnostic Notification Function

Member `srdia_UpdateConnectionDiagnostics` (const uint32_t connection_id, const uint32_t round_trip_delay, const uint32_t alive_time)
RASW-645 Update Connection Diagnostics Function

Member `srmd4_CalculateMd4` (const srcty_Md4InitValue md4_initial_value, const uint16_t data_size, const uint8_t *const data_buffer, srmd4_Md4 *const calculated_md4)
RASW-633 Calculate MD4 Function
RASW-634 Safety Code

Member `srmsg_CheckMessage` (const srtyp_SrMessage *const sr_message)
RASW-160 General PDU Message Structure
RASW-616 Check Message Function

Member `srmsg_CreateConnReqMessage` (const srtyp_SrMessageHeaderCreate message_header, const srtyp_ProtocolVersion protocol_version, const uint16_t n_send_max, srtyp_SrMessage *const sr_message)
RASW-617 Create Connection Request Message Function
RASW-170 Connection Request Message Structure
RASW-172 Message Length
RASW-173 Protocol Version
RASW-174 Nsendmax
RASW-175 Reserve

Member `srmsg_CreateConnRespMessage` (const srtyp_SrMessageHeaderCreate message_header, const srtyp_ProtocolVersion protocol_version, const uint16_t n_send_max, srtyp_SrMessage *const sr_message)
RASW-618 Create Connection Response Message Function
RASW-171 Connection Response Message Structure
RASW-172 Message Length
RASW-173 Protocol Version
RASW-174 Nsendmax
RASW-175 Reserve

Member `srmsg_CreateDataMessage` (const srtyp_SrMessageHeaderCreate message_header, const srtyp_SrMessagePayload *const message_payload, srtyp_SrMessage *const sr_message)
RASW-623 Create Data Message Function
RASW-191 Data Message Structure
RASW-192 Message Length
RASW-193 Size of Message Data
RASW-194 Data Message

Member `srmsg_CreateDiscReqMessage` (const srtyp_SrMessageHeaderCreate message_header, const uint16_t detailed_reason, const sraty_DiscReason reason, srtyp_SrMessage *const sr_message)
RASW-621 Create Disconnection Request Message Function
RASW-183 Disconnection Request Message Structure
RASW-184 Message Length
RASW-185 Detailed Information regarding the Reason for the Disconnection Request
RASW-186 Reason for Disconnection Request

Member `srmsg_CreateHeartbeatMessage` (const srtyp_SrMessageHeaderCreate message_header, srtyp_SrMessage *const sr_message)
RASW-622 Create Heartbeat Message Function
RASW-188 Heartbeat Message Structure
RASW-189 Message Length

Member `srmsg_CreateRetrDataMessage` (const `srtyp_SrMessageHeaderCreate` `message_header`, const `srtyp_SrMessagePayload` *const `message_payload`, `srtyp_SrMessage` *const `sr_message`)

RASW-624 Create Retransmitted Data Message Function
RASW-196 Retransmitted Data Message Structure
RASW-192 Message Length
RASW-193 Size of Message Data
RASW-194 Data Message

Member `srmsg_CreateRetrReqMessage` (const `srtyp_SrMessageHeaderCreate` `message_header`, `srtyp_SrMessage` *const `sr_message`)

RASW-619 Create Retransmission Request Message Function
RASW-177 Retransmission Request Message Structure
RASW-178 Message Length

Member `srmsg_CreateRetrRespMessage` (const `srtyp_SrMessageHeaderCreate` `message_header`, `srtyp_SrMessage` *const `sr_message`)

RASW-620 Create Retransmission Response Message Function
RASW-180 Retransmission Response Message Structure
RASW-178 Message Length

Member `srmsg_GetConnMessageData` (const `srtyp_SrMessage` *const `sr_message`, `srtyp_ProtocolVersion` *const `protocol_version`, `uint16_t` *const `n_send_max`)

RASW-625 Get Connection Message Data Function
RASW-173 Protocol Version
RASW-174 Nsendmax

Member `srmsg_GetDataMessagePayload` (const `srtyp_SrMessage` *const `sr_message`, *const `srtyp_SrMessagePayload` *const `message_payload`)

RASW-626 Get Data Message Payload Function
RASW-193 Size of Message Data
RASW-194 Data Message

Member `srmsg_GetDiscMessageData` (const `srtyp_SrMessage` *const `sr_message`, `uint16_t` *const `detailed_reason`, `srtyp_DiscReason` *const `reason`)

RASW-627 Get Disconnection Message Data Function
RASW-185 Detailed Information regarding the Reason for the Disconnection Request
RASW-186 Reason for Disconnection Request

Member `srmsg_GetMessageHeader` (const `srtyp_SrMessage` *const `sr_message`, `srtyp_SrMessageHeader` *const `message_header`)

RASW-628 Get Message Header Function
RASW-160 General PDU Message Structure
RASW-161 Message Type
RASW-162 Receiver Identification
RASW-163 Sender Identification
RASW-164 Sequence Number
RASW-165 Confirmed Sequence Number
RASW-166 Time Stamp
RASW-167 Confirmed Time Stamp

Member `srmsg_GetMessageSequenceNumber` (const `srtyp_SrMessage` *const `sr_message`)

RASW-825 Get Message Sequence Number Function

Member `srmsg_GetMessageType` (const `srtyp_SrMessage` *const `sr_message`)

RASW-824 Get Message Type Function

Member `srmsg_Init` (const `srcty_SafetyCodeType configured_safety_code_type, const srcty_Md4InitValue configured_md4_initial_value)`

RASW-629 Init sr_messages Function

Member `srmsg_UpdateMessageHeader` (const `srtyp_SrMessageHeaderUpdate message_header_update, srtyp_SrMessage *const sr_message)`

RASW-630 Update Message Header Function

RASW-164 Sequence Number

RASW-165 Confirmed Sequence Number

RASW-166 Time Stamp

RASW-168 Safety Code

Member `srnot_ConnectionStateNotification` (const `uint32_t connection_id, const sraty_ConnectionStates connection_state, const sraty_BufferUtilisation buffer_utilisation, const uint16_t opposite_buffer_size, const sraty_DiscReason disconnect_reason, const uint16_t detailed_disconnect_reason)`

RASW-555 Connection State Notification

RASW-296 Connection State Notification Structure

RASW-299 Connection Identification

RASW-298 Connection State

RASW-293 Buffer Utilisation

RASW-291 Opposite Buffer Size

RASW-295 Disconnect Reason

RASW-294 Detailed Disconnect Reason

Member `srnot_MessageReceivedNotification` (const `uint32_t connection_id)`

RASW-554 Message Received Notification

RASW-279 Message Received Notification Structure

RASW-302 Connection Identification

Member `srnot_RedDiagnosticNotification` (const `uint32_t connection_id, const sraty_RedundancyChannelDiagnosticData redundancy_channel_diagnostic_data)`

RASW-557 Red Diagnostic Notification

RASW-325 RedL Diagnostic Notification Structure

RASW-323 Connection Identification

RASW-315 Redundancy Channel Diagnostic Data

Member `srnot_SrDiagnosticNotification` (const `uint32_t connection_id, const sraty_ConnectionDiagnosticData connection_diagnostic_data)`

RASW-556 Sr Diagnostic Notification

RASW-327 SafRetL Diagnostic Notification Structure

RASW-326 Connection Identification

RASW-321 Connection Diagnostic Data

Member `srrece_AddToBuffer` (const `uint32_t connection_id, const srtyp_SrMessagePayload *const message_payload)`

RASW-608 Add to Buffer Function

Member `srrece_GetFreeBufferEntries` (const `uint32_t connection_id)`

RASW-610 Get Free Buffer Entries Function

Member `srrece_GetPayloadSizeOfNextMessageToRead` (const `uint32_t connection_id)`

RASW-823 Get Payload Size of Next Message To Read Function

Member `srrece_GetUsedBufferEntries` (const `uint32_t connection_id)`

RASW-609 Get Used Buffer Entries Function

Member `srrece_Init` (const uint32_t configured_connections, const uint16_t configured_n_send_max)
RASW-611 Init sr_received_buffer Function

Member `srrece_InitBuffer` (const uint32_t connection_id)
RASW-612 Init Buffer Function

Member `srrece_ReadFromBuffer` (const uint32_t connection_id, srtyp_SrMessagePayload *const message_payload)
RASW-613 Read from Buffer Function

Member `srsend_AddToBuffer` (const uint32_t connection_id, const srtyp_SrMessage *const message)
RASW-596 Add to Buffer Function

Member `srsend_GetFreeBufferEntries` (const uint32_t connection_id)
RASW-598 Get Free Buffer Entries Function

Member `srsend_GetNumberOfMessagesToSend` (const uint32_t connection_id)
RASW-599 Get Number of Messages to Send Function

Member `srsend_GetUsedBufferEntries` (const uint32_t connection_id)
RASW-597 Get Used Buffer Entries Function

Member `srsend_Init` (const uint32_t configured_connections)
RASW-600 Init sr_send_buffer Function

Member `srsend_InitBuffer` (const uint32_t connection_id)
RASW-601 Init Buffer Function

Member `srsend_IsSequenceNumberInBuffer` (const uint32_t connection_id, const uint32_t sequence_number)
RASW-602 Is Sequence Number in Buffer Function

Member `srsend_PrepateBufferForRetr` (const uint32_t connection_id, const uint32_t sequence_number_for_retransmission, const srtyp_SrMessageHeaderCreate message_header, uint32_t *const new_current_sequence_number)
RASW-603 Prepare Buffer for Retransmission Function

Member `srsend_ReadMessageToSend` (const uint32_t connection_id, srtyp_SrMessage *const message)
RASW-604 Read Message to Send Function

Member `srsend_RemoveFromBuffer` (const uint32_t connection_id, const uint32_t confirmed_sequence_number)
RASW-605 Remove from Buffer Function

Member `srstm_GetConnectionState` (const uint32_t connection_id)
RASW-561 Get Connection State Function

Member `srstm_Init` (const uint32_t configured_connections)
RASW-562 Init sr_state_machine Function

Member `srstm_ProcessConnectionStateMachine` (const uint32_t connection_id, const srtyp_ConnectionEvents event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
RASW-563 Process Connection State Machine Function
RASW-560 sr_state_machine Events
RASW-749 Connection State Notification Sequence
RASW-755 Process State Machine Receive ConnReq Message Sequence
RASW-757 Process State Machine Receive ConnResp Message Sequence
RASW-759 Process State Machine Receive Data Message (in Sequence) Sequence
RASW-761 Process State Machine Receive RetrReq Message (in Sequence) Sequence
RASW-763 Process State Machine Receive RetrReq Message (not in Sequence) Sequence
RASW-765 Send Pending Messages Sequence

Member [srtyp_ConnectionEvents](#)

RASW-560 sr_state_machine Events

Member [StartRetransmission](#) (const uint32_t connection_id, const sraty_ConnectionStates new_state, bool retransmission_requested)

RASW-563 Process Connection State Machine Function

Member [Step](#) (const Md4Function md4_function, uint32_t *const a, const uint32_t b, const uint32_t c, const uint32_t d, const uint32_t x, const uint32_t s)

RASW-633 Calculate MD4 Function

Member [UpdateConnectionState](#) (const uint32_t connection_id, const sraty_ConnectionStates new_state)

RASW-563 Process Connection State Machine Function

RASW-749 Connection State Notification Sequence

Member [UpdateConnectionStateWithDiscReason](#) (const uint32_t connection_id, const sraty_ConnectionStates new_state, const sraty_DiscReason disconnect_reason)

RASW-563 Process Connection State Machine Function

RASW-749 Connection State Notification Sequence

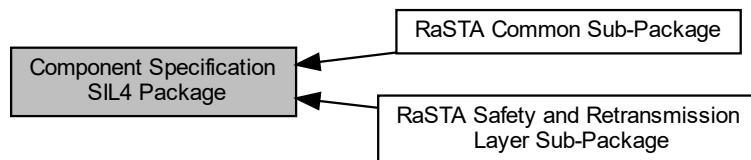
Member [WriteU32toByteArray](#) (const uint32_t source, uint8_t *const destination)

RASW-633 Calculate MD4 Function

4 Module Documentation

4.1 Component Specification SIL4 Package

Collaboration diagram for Component Specification SIL4 Package:



Modules

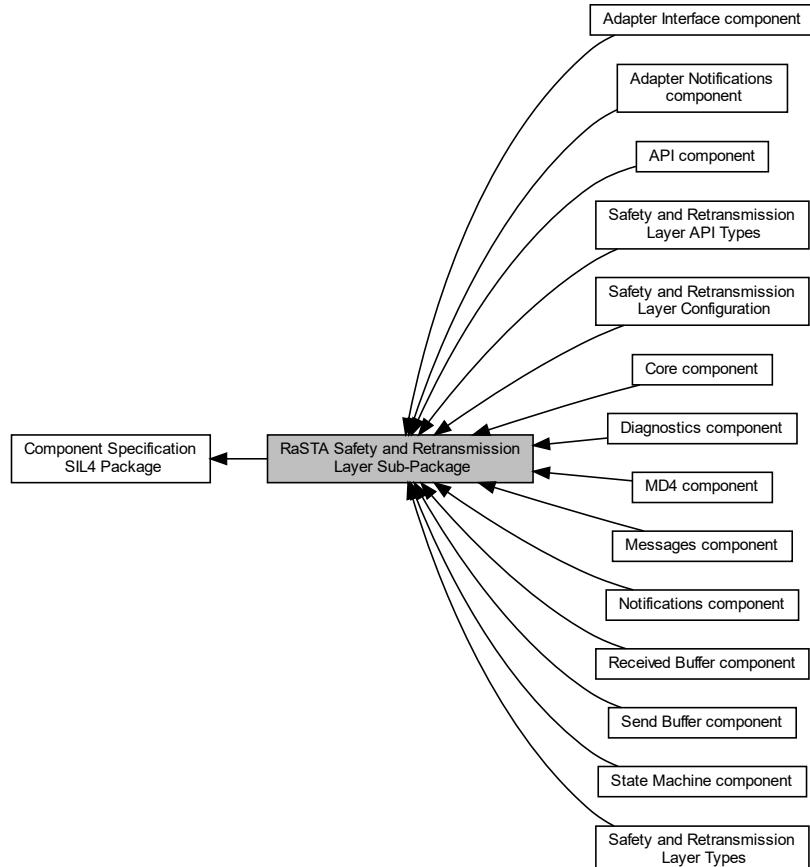
- [RaSTA Safety and Retransmission Layer Sub-Package](#)
- [RaSTA Common Sub-Package](#)

4.1.1 Detailed Description

Specification of all safety related components of the RaSTA Stack.

4.2 RaSTA Safety and Retransmission Layer Sub-Package

Collaboration diagram for RaSTA Safety and Retransmission Layer Sub-Package:



Modules

- API component
Interface of RaSTA SafRetL API.
- Notifications component
Interface of RaSTA SafRetL notifications to the upper layer.
- State Machine component
Interface of RaSTA SafRetL state machine module.
- Core component
Interface of RaSTA SafRetL core module.
- Send Buffer component
Interface of RaSTA SafRetL send buffer module.
- Received Buffer component
Interface of RaSTA SafRetL received buffer module.
- Messages component
Interface of RaSTA SafRetL messages module.
- MD4 component

- *Interface of RaSTA SafRetL MD4 module.*
- **Diagnostics component**
Interface of RaSTA SafRetL diagnostics module.
- **Adapter Interface component**
Interface of RaSTA SafRetL adapter.
- **Adapter Notifications component**
Interface of RaSTA SafRetL adapter notifications.
- **Safety and Retransmission Layer Configuration**
Type definitions of RaSTA SafRetL configuration.
- **Safety and Retransmission Layer API Types**
Type definitions of RaSTA SafRetL API.
- **Safety and Retransmission Layer Types**
Internal type definitions of RaSTA SafRetL.

4.2.1 Detailed Description

Specification of all components within the RaSTA Safety and Retransmission Layer.

4.3 API component

Interface of RaSTA SafRetL API.

Collaboration diagram for API component:



Functions

- `radef_RaStaReturnCode srapi_Init (const srcty_SafetyRetransmissionConfiguration *const safety_retransmission_configuration)`
Initialize SafRetL.
- `radef_RaStaReturnCode srapi_GetInitializationState (void)`
Get the initialization state of the SafRetL.
- `radef_RaStaReturnCode srapi_OpenConnection (const uint32_t sender_id, const uint32_t receiver_id, const uint32_t network_id, uint32_t *const connection_id)`
Open a RaSTA connection.
- `radef_RaStaReturnCode srapi_CloseConnection (const uint32_t connection_id, const uint16_t detailed_reason)`
Close a RaSTA connection.
- `radef_RaStaReturnCode srapi_SendData (const uint32_t connection_id, const uint16_t message_size, const uint8_t *const message_data)`
Send a RaSTA data message.

- `radef_RaStaReturnCode srapi_ReadData` (const uint32_t connection_id, const uint16_t buffer_size, uint16_t *const message_size, uint8_t *const message_buffer)
Read the data of a received RaSTA message.
- `radef_RaStaReturnCode srapi_GetConnectionState` (const uint32_t connection_id, `srtty_ConnectionStates` *const connection_state, `srtty_BufferUtilisation` *const buffer_utilisation, uint16_t *const opposite_buffer_size)
Get the state of a connection.
- `radef_RaStaReturnCode srapi_CheckTimings` (void)
Check SafRetL timings.

4.3.1 Detailed Description

Interface of RaSTA SafRetL API.

Specification of the API component of the RaSTA Safety and Retransmission Layer.

This module defines and implements the interface functions (like get initialization state, open & close connection, send & read data, get connection state) for the application layer, as they act as entry point to use the RaSTA protocol stack.

Implements Requirements [RASW-543](#) Component sr_api Overview

[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

[RASW-520](#) Error Handling

[RASW-521](#) Input Parameter Check

4.3.2 Function Documentation

4.3.2.1 `srapi_CheckTimings()` `radef_RaStaReturnCode srapi_CheckTimings (void)`

Check SafRetL timings.

This function is used to check the timings of the SafRetL. If the sr_api module is not initialized, a `radef_kNotInitialized` error is returned. It must be called periodically. It has 3 main uses:

- Received messages polling (read all available messages from the adapter layer and process them in the state machine). Read and process new messages while `srcor_IsReceivedMsgPendingAndBuffersNotFull` check return true.
- Send pending messages (send not yet send messages from the send buffer)
- Check timings for message timeout & HB interval
 - Check with `srcor_IsMessageTimeout` if a message timeout occurred. If true, send a `srtyp_kConnEventTimeout` event to the state machine.
 - Check with `srcor_IsHeartbeatInterval` if the heartbeat interval elapsed and also if there are not send messages in the send buffer. If true, send a `srtyp_kConnEventSendHb` event to the state machine.

These checks are done for all available connections.

Implements Requirements

- RASW-551 Check Timings Function
- RASW-319 Check Timings Function Structure
- RASW-317 Error Code
- RASW-503 Enum RaSta Return Code Usage
- RASW-753 Check Timings Sequence
- RASW-769 Received Message Polling Sequence
- RASW-773 Check Timings Message Timeout / Heartbeat Sequence

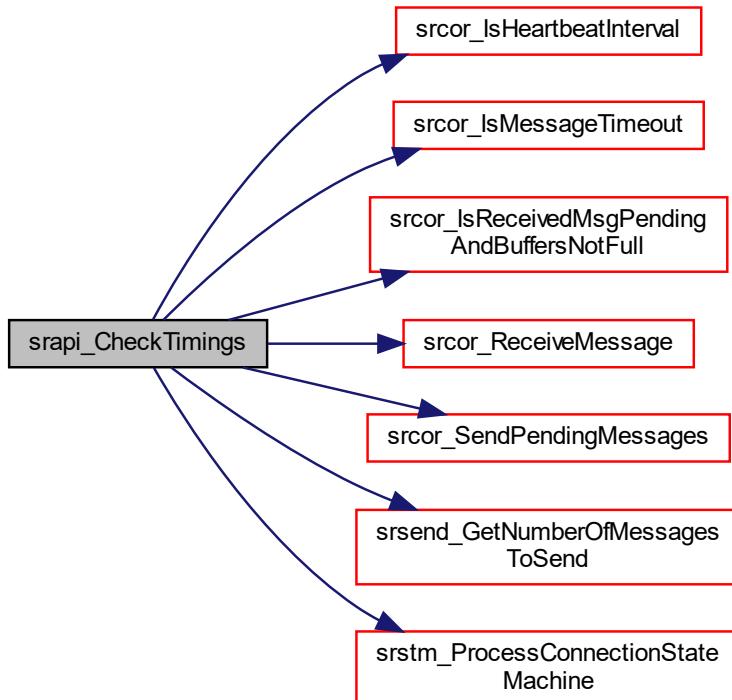
Remarks

This function must be called periodically, in an appropriate interval related to the configured timings.

Returns

- radef_kNoError -> successful operation
- radef_kNotInitialized -> module not initialized

Here is the call graph for this function:



4.3.2.2 `srapi_CloseConnection()` `radef_RaStaReturnCode srapi_CloseConnection (`
`const uint32_t connection_id,`
`const uint16_t detailed_reason)`

Close a RaSTA connection.

This function is used to close a specific RaSTA connection. If the sr_api module is not initialized, a `radef_kNotInitialized` error is returned. If initialized, the connection id is checked if it is in a valid range, otherwise a `radef_kInvalidParameter` is returned. If everything is good, the detailed disconnection reason is set with the `srcor_SetDiscDetailedReason` function. Then the `srstm_ProcessConnectionStateMachine` function is called with `srtyp_kConnEventClose` event to close the Rasta connection.

Implements Requirements [RASW-547](#) Close Connection Function
[RASW-318](#) Close Connection Function Structure
[RASW-316](#) Connection Identification
[RASW-309](#) Detailed Reason
[RASW-308](#) Error Code
[RASW-503](#) Enum RaSta Return Code Usage
[RASW-739](#) Close Connection Sequence

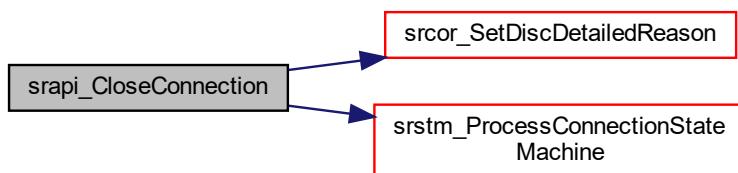
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<code>detailed_reason</code>	Detailed reason for disconnection. The full value range is valid and usable.

Returns

`radef_kNoError` -> successful operation
`radef_kNotInitialized` -> module not initialized
`radef_kInvalidParameter` -> invalid parameter

Here is the call graph for this function:



```
4.3.2.3 srapi_GetConnectionState() radef_RaStaReturnCode srapi_GetConnectionState (
    const uint32_t connection_id,
    sraty_ConnectionStates *const connection_state,
    sraty_BufferUtilisation *const buffer_utilisation,
    uint16_t *const opposite_buffer_size )
```

Get the state of a connection.

This function is used to get the connection state of a specific RaSTA connection. If the sr_api module is not initialized, a `radef_kNotInitialized` error is returned. If initialized, the connection id is checked if it is in a valid range, otherwise a `radef_kInvalidParameter` is returned. If everything is good, the connection state is obtained from the state machine with the `srstm_GetConnectionState` function call. With the `srcor_GetBufferSizeAndUtilisation` further information is collected to then be returned to the caller.

Implements Requirements

- [RASW-550](#) Get Connection State Function
- [RASW-288](#) Get Connection State Function Structure
- [RASW-287](#) Connection Identification
- [RASW-282](#) Connection State
- [RASW-281](#) Buffer Utilisation
- [RASW-284](#) Opposite Buffer Size
- [RASW-283](#) Error Code
- [RASW-503](#) Enum RaSta Return Code Usage
- [RASW-745](#) Get Connection State Sequence

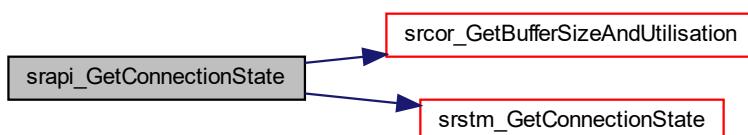
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
out	<code>connection_state</code>	State of the connection. If the pointer is NULL, a <code>radef_kInvalidParameter</code> error is returned.
out	<code>buffer_utilisation</code>	Struct with own buffer utilisation data. If the pointer is NULL, a <code>radef_kInvalidParameter</code> error is returned.
out	<code>opposite_buffer_size</code>	Size of the receive buffer of the connection party. If the pointer is NULL, a <code>radef_kInvalidParameter</code> error is returned.

Returns

[radef_kNoError](#) -> successful operation
[radef_kNotInitialized](#) -> module not initialized
[radef_kInvalidParameter](#) -> invalid parameter

Here is the call graph for this function:



4.3.2.4 srapi_GetInitializationState() `radef_RaStaReturnCode srapi_GetInitializationState (void)`

Get the initialization state of the SafRetL.

This function is used to check the initialization state of the sr_api module.

Implements Requirements [RASW-545](#) Get Initialization State Function
[RASW-306](#) Get Initialization State Function Structure
[RASW-305](#) Error Code
[RASW-503](#) Enum RaSta Return Code Usage

Returns

`radef_kNoError` -> successful initialized
`radef_kNotInitialized` -> not initialized

Here is the caller graph for this function:



4.3.2.5 srapi_Init() `radef_RaStaReturnCode srapi_Init (const srcty_SafetyRetransmissionConfiguration *const safety_retransmission_configuration)`

Initialize SafRetL.

This function is used to initialize the sr_api module. If the sr_api module is already initialized, a `radef_kAlreadyInitialized` error is returned. After checking the configuration for validity, the pointer to the configuration is saved internally. If it is not valid, a `radef_kInvalidConfiguration` is returned. The configuration is then used to initialize the state machine and the core module.

Implements Requirements [RASW-544](#) Init sr_api Function
[RASW-267](#) Initialization Function Structure
[RASW-292](#) Configuration SafRetL
[RASW-290](#) Error Code
[RASW-503](#) Enum RaSta Return Code Usage
[RASW-735](#) Init SafRetL Sequence

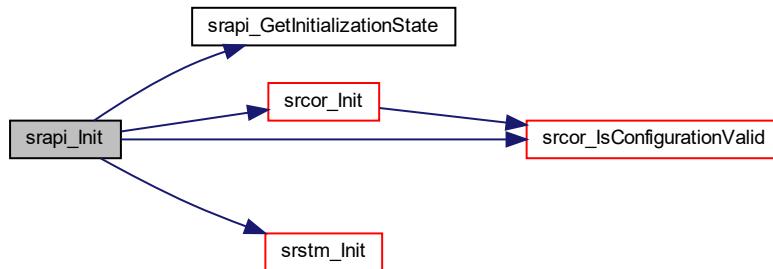
Parameters

in	<i>safety_retransmission_configuration</i>	Pointer to SafRetL configuration. More details about valid configuration can be found directly in srcy_SafetyRetransmissionConfiguration . If the pointer is NULL, a radef_kInvalidParameter error is returned.
----	--	---

Returns

[radef_kNoError](#) -> successful initialized
[radef_kAlreadyInitialized](#) -> module already initialized
[radef_kInvalidConfiguration](#) -> invalid configuration data
[radef_kInvalidParameter](#) -> invalid parameter

Here is the call graph for this function:

**4.3.2.6 srapi_OpenConnection()** [radef_RaStaReturnCode](#) srapi_OpenConnection (

```

    const uint32_t sender_id,
    const uint32_t receiver_id,
    const uint32_t network_id,
    uint32_t *const connection_id )
  
```

Open a RaSTA connection.

This function is used to open a specific RaSTA connection. If the sr_api module is not initialized, a [radef_kNotInitialized](#) error is returned. The network id is checked if it is known in the saved configuration. If it is not known, a [radef_kInvalidParameter](#) is returned. Afterwards the connection id is obtained by calling the [srcor_GetConnectionId](#) function. If no valid connection is found, a [radef_kInvalidParameter](#) is returned. If all tests passed, the [srstm_ProcessConnectionStateMachine](#) function is called with an [srtyp_kConnEventOpen](#) event. The connection id is passed to the application via the output parameter.

Implements Requirements

[RASW-546](#) Open Connection Function

[RASW-303](#) Open Connection Function Structure

[RASW-301](#) Sender Identification

RASW-300 Receiver Identification
RASW-324 Network Identification
RASW-322 Connection Identification
RASW-314 Error Code
RASW-503 Enum RaSta Return Code Usage
RASW-737 Open Connection Sequence

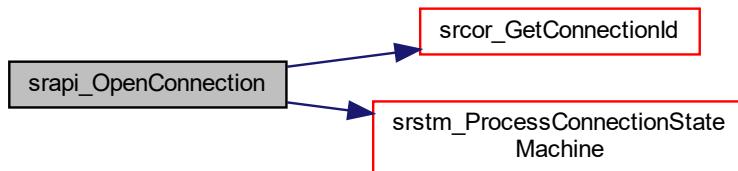
Parameters

in	<i>sender_id</i>	RaSTA sender identification. The full value range is valid and usable.
in	<i>receiver_id</i>	RaSTA receiver identification. The full value range is valid and usable.
in	<i>network_id</i>	RaSTA network identification. The full value range is valid and usable.
out	<i>connection_id</i>	Connection identification for the opened connection. If the pointer is NULL, a radef_kInvalidParameter error is returned.

Returns

[radef_kNoError](#) -> successful operation
[radef_kNotInitialized](#) -> module not initialized
[radef_kInvalidParameter](#) -> invalid parameter

Here is the call graph for this function:



4.3.2.7 srapi_ReadData() [radef_RaStaReturnCode](#) srapi_ReadData (

```

const uint32_t connection_id,
const uint16_t buffer_size,
uint16_t *const message_size,
uint8_t *const message_buffer )
  
```

Read the data of a received RaSTA message.

This function is used to read data from a specific RaSTA connection. If the sr_api module is not initialized, a [radef_kNotInitialized](#) error is returned. If initialized, the connection id is checked if it is in a valid range, otherwise a [radef_kInvalidParameter](#) is returned. If the provided buffer size is not big enough for the next message from the received buffer, a [radef_kInvalidBufferSize](#) is returned. If everything is good, a message is read with [srrece_ReadFromBuffer](#). If no message is available, a [radef_kNoMessageReceived](#) is returned. Otherwise the received data message is copied into the provided parameters (message_size & message_data).

Implements Requirements

- RASW-549 Read Data Function
- RASW-274 Read Data Function Structure
- RASW-273 Connection Identification
- RASW-269 Buffer Size
- RASW-268 Message Size
- RASW-270 Message Buffer
- RASW-286 Error Code
- RASW-503 Enum RaSta Return Code Usage
- RASW-743 Read Data Sequence

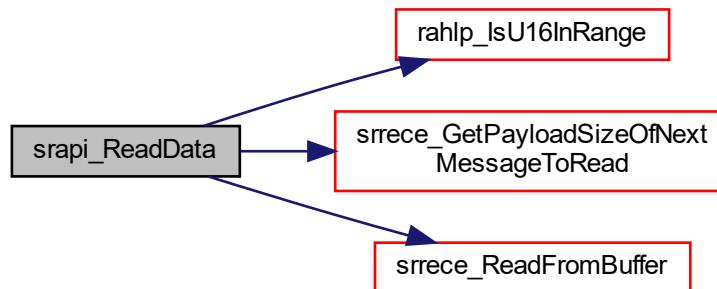
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>buffer_size</i>	Size of the buffer provided to parameter <i>message_buffer</i> [bytes]. Valid range: <code>srcty_kMinSrLayerPayloadDataSize</code> $\leq \text{value} \leq \text{RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE}$. Any value in this range can be used, must be large enough to store the received message.
out	<i>message_size</i>	Pointer to the size of the received message data [bytes]. If the pointer is NULL, a <code>radef_kInvalidParameter</code> error is returned.
out	<i>message_buffer</i>	Pointer to a buffer for saving the received message. If the pointer is NULL, a <code>radef_kInvalidParameter</code> error is returned.

Returns

- `radef_kNoError` -> successful operation
- `radef_kNoMessageReceived` -> no message received (used for polling)
- `radef_kNotInitialized` -> module not initialized
- `radef_kInvalidParameter` -> invalid parameter
- `radef_kInvalidBufferSize` -> invalid buffer size

Here is the call graph for this function:



4.3.2.8 `srapi_SendData()` `radef_RaStaReturnCode srapi_SendData (`
 `const uint32_t connection_id,`
 `const uint16_t message_size,`
 `const uint8_t *const message_data)`

Send a RaSTA data message.

This function is used to send data over a specific RaSTA connection. If the sr_api module is not initialized, a `radef_kNotInitialized` error is returned. If initialized, the connection id is checked if it is in a valid range, otherwise a `radef_kInvalidParameter` is returned. If the state machine is in closed state, a `radef_kInvalidOperationInCurrentState` is returned, since no data can be send when the connection is closed. If everything is good, it is checked if there are free entries in the send buffer with the `srsend_GetFreeBufferEntries` function. In case the buffer is full, a `radef_kSendBufferFull` error is returned. If there is space, the data payload is buffered with the `srcor_WriteMessagePayloadToTemporaryBuffer` function before calling the `srstm_ProcessConnectionStateMachine` function with an `srtyp_kConnEventSendData` event to process the buffered data message and send it to the connection party.

Implements Requirements [RASW-548](#) Send Data Function
[RASW-310](#) Send Data Function Structure
[RASW-275](#) Connection Identification
[RASW-277](#) Message Size
[RASW-276](#) Message Data
[RASW-272](#) Error Code
[RASW-503](#) Enum RaSta Return Code Usage
[RASW-741](#) Send Data Sequence

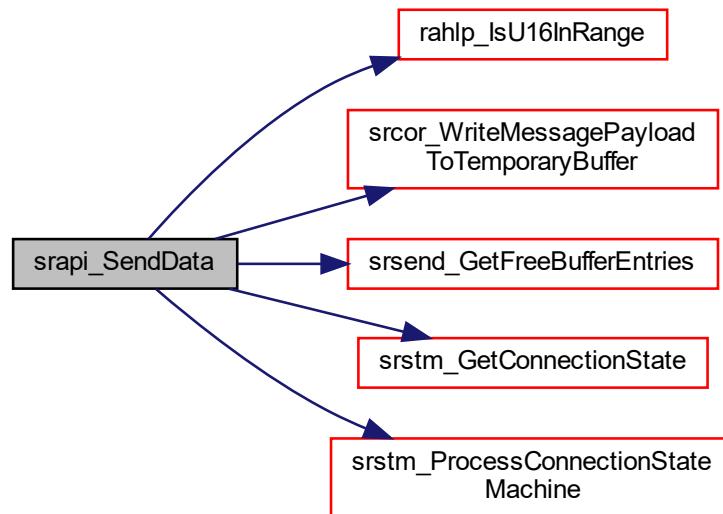
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>message_size</i>	Size of the message data [bytes]. Valid range: <code>srcty_kMinSrLayerPayloadDataSize</code> $\leq \text{value} \leq \text{RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE}$. If the value is outside this range, a <code>radef_kInvalidMessageSize</code> error is returned.
in	<i>message_data</i>	Pointer to message data array. If the pointer is NULL, a <code>radef_kInvalidParameter</code> error is returned. For the message data the full value range is valid and usable.

Returns

`radef_kNoError` -> successful operation
`radef_kNotInitialized` -> module not initialized
`radef_kInvalidParameter` -> invalid parameter
`radef_kInvalidMessageSize` -> invalid message size
`radef_kSendBufferFull` -> send buffer full
`radef_kInvalidOperationInCurrentState` -> state machine in closed state

Here is the call graph for this function:



4.4 Notifications component

Interface of RaSTA SafRetL notifications to the upper layer.

Collaboration diagram for Notifications component:



Functions

- void [srnot_MessageReceivedNotification](#) (const uint32_t connection_id)
SafRetL message received notification function.
 - void [srnot_ConnectionStateNotification](#) (const uint32_t connection_id, const [sraty_ConnectionStates](#) connection_state, const [sraty_BufferUtilisation](#) buffer_utilisation, const uint16_t opposite_buffer_size, const [sraty_DiscReason](#) disconnect_reason, const uint16_t detailed_disconnect_reason)
SafRetL connection state notification function.
 - void [srnot_SrDiagnosticNotification](#) (const uint32_t connection_id, const [sraty_ConnectionDiagnosticData](#) connection_diagnostic_data)
SafRetL diagnostic data notification function.
 - void [srnot_RedDiagnosticNotification](#) (const uint32_t connection_id, const [sraty_RedundancyChannelDiagnosticData](#) redundancy_channel_diagnostic_data)
Forwarded RedL diagnostic data notification function.

4.4.1 Detailed Description

Interface of RaSTA SafRetL notifications to the upper layer.

Specification of the Notifications component of the RaSTA Safety and Retransmission Layer.

This module defines the notification functions (like message received, connection state and diagnostic) for the application layer. The SafRetL only defines the interface, the implementation of this notification functions must be done in the application layer.

Implements Requirements [RASW-553](#) Component sr_notifications Overview
[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

4.4.2 Function Documentation

4.4.2.1 srnot_ConnectionStateNotification() `void srnot_ConnectionStateNotification (`
 `const uint32_t connection_id,`
 `const sraty_ConnectionStates connection_state,`
 `const sraty_BufferUtilisation buffer_utilisation,`
 `const uint16_t opposite_buffer_size,`
 `const sraty_DiscReason disconnect_reason,`
 `const uint16_t detailed_disconnect_reason)`

SafRetL connection state notification function.

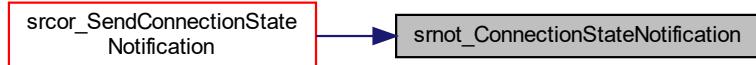
This function is called by the SafRetL to notify the application layer about a connection state change of a specific connection. Additionally many informations about the specified connection are send with this notification.

Implements Requirements [RASW-555](#) Connection State Notification
[RASW-296](#) Connection State Notification Structure
[RASW-299](#) Connection Identification
[RASW-298](#) Connection State
[RASW-293](#) Buffer Utilisation
[RASW-291](#) Opposite Buffer Size
[RASW-295](#) Disconnect Reason
[RASW-294](#) Detailed Disconnect Reason

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>connection_state</i>	Current connection state of the SafRetL. Valid range: sraty_kConnectionMin $\leq \text{value} < \text{sraty_kConnectionMax}$.
in	<i>buffer_utilisation</i>	Struct with buffer utilisation data. Structure and valid ranges can be found in sraty_BufferUtilisation .
in	<i>opposite_buffer_size</i>	Size of the receive buffer of the opposite RaSTA instance. Full value range is valid and usable.
in	<i>disconnect_reason</i>	Disconnect reason (only valid if connection state changed to sraty_kConnectionClosed). Valid range: sraty_kDiscReasonMin $\leq \text{value} < \text{sraty_kDiscReasonMax}$.
in	<i>detailed_disconnect_reason</i>	Detailed disconnect reason application (only valid if connection state changed to sraty_kConnectionClosed). Full value range is valid and usable.

Here is the caller graph for this function:



4.4.2.2 srnot_MessageReceivedNotification() `void srnot_MessageReceivedNotification (const uint32_t connection_id)`

SafRetL message received notification function.

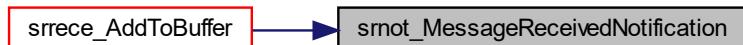
This function is called by the SafRetL to notify the application layer that a received message from a specific connection is ready to be read.

Implements Requirements [RASW-554](#) Message Received Notification
[RASW-279](#) Message Received Notification Structure
[RASW-302](#) Connection Identification

Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
----	----------------------------	---

Here is the caller graph for this function:



4.4.2.3 srnot_RedDiagnosticNotification() `void srnot_RedDiagnosticNotification (const uint32_t connection_id, const sraty_RedundancyChannelDiagnosticData redundancy_channel_diagnostic_data)`

Forwarded RedL diagnostic data notification function.

This function is called by the SafRetL to notify the application layer about new diagnostic data from the RedL. This data is forwarded without any modification. This contains the transport channel identification, diagnostic window size, number of missed messages and the average delay indicators Tdrift und Tdrift2.

Implements Requirements	RASW-557 Red Diagnostic Notification RASW-325 RedL Diagnostic Notification Structure RASW-323 Connection Identification RASW-315 Redundancy Channel Diagnostic Data
--------------------------------	--

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
in	<i>redundancy_channel_diagnostic_data</i>	Diagnostic data from the redundancy channel. Structure and valid ranges can be found in sraty_RedundancyChannelDiagnosticData .

Here is the caller graph for this function:



4.4.2.4 srnot_SrDiagnosticNotification() void srnot_SrDiagnosticNotification (const uint32_t *connection_id*, const [sraty_ConnectionDiagnosticData](#) *connection_diagnostic_data*)

SafRetL diagnostic data notification function.

This function is called by the SafRetL to notify the application layer about new diagnostic data from the SafRetL. This contains all error counters as well as the distribution of the round trip delay time and the alive time.

Implements Requirements	RASW-556 Sr Diagnostic Notification RASW-327 SafRetL Diagnostic Notification Structure RASW-326 Connection Identification RASW-321 Connection Diagnostic Data
--------------------------------	--

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
in	<i>connection_diagnostic_data</i>	Diagnostic data of the SafRetL. Structure and valid ranges can be found in sraty_ConnectionDiagnosticData .

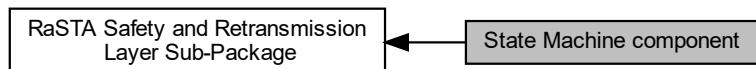
Here is the caller graph for this function:



4.5 State Machine component

Interface of RaSTA SafRetL state machine module.

Collaboration diagram for State Machine component:



Functions

- static void `ProcessStateClosedEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event)
Process events in state Closed.
- static void `ProcessStateDownEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event)
Process events in state Down.
- static void `ProcessStateStartEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process events in state Start.
- static void `ProcessStateUpEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process events in state Up.
- static void `ProcessStateRetransRequestEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq)
Process events in state RetransRequest.
- static void `ProcessStateRetransRunningEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process events in state RetransRunning.
- static bool `ProcessReceivedMessage` (const uint32_t connection_id)
Process received message and evaluate timeliness.
- static void `CloseConnection` (const uint32_t connection_id, const `srtyp_DiscReason` disconnect_reason, const bool is_incoming_message)
Close the connection of a specific RaSTA connection.
- static void `CloseRedundancyChannel` (const uint32_t connection_id, const bool is_incoming_message)
Close the redundancy channel of a specific RaSTA connection.

- static void [StartRetransmission](#) (const uint32_t connection_id, const [sraty_ConnectionStates](#) new_state, bool retransmission_requested)
Start a retransmission.
- static void [UpdateConnectionState](#) (const uint32_t connection_id, const [sraty_ConnectionStates](#) new_state)
Update the connection state and send a connection state notification to the application layer.
- static void [UpdateConnectionStateWithDiscReason](#) (const uint32_t connection_id, const [sraty_ConnectionStates](#) new_state, const [sraty_DiscReason](#) disconnect_reason)
Update the connection state and send a connection state notification to the application layer.
- void [srstm_Init](#) (const uint32_t configured_connections)
Initialize SafRetL state machine module.
- void [srstm_ProcessConnectionStateMachine](#) (const uint32_t connection_id, const [srtp_ConnectionEvents](#) event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process RaSTA connection state machine.
- [sraty_ConnectionStates srstm_GetConnectionState](#) (const uint32_t connection_id)
Return the state of a RaSTA connection state machine.

4.5.1 Detailed Description

Interface of RaSTA SafRetL state machine module.

Specification of the State Machine component of the RaSTA Safety and Retransmission Layer.

This module provides all needed functionality to reflect the logic of the state machine. The state machine reacts on:

- occurring events
- function calls from the application layer
- notifications from the SafRetL adapter layer

The state machine evaluates them, initiates the necessary actions and changes the state.

Implements Requirements [RASW-559](#) Component sr_state_machine Overview
[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level
[RASW-520](#) Error Handling
[RASW-521](#) Input Parameter Check

4.5.2 Function Documentation

4.5.2.1 CloseConnection() static void CloseConnection (
 const uint32_t connection_id,
 const [sraty_DiscReason](#) disconnect_reason,
 const bool is_incoming_message) [static]

Close the connection of a specific RaSTA connection.

This internal function is used to close a connection of a specific connection. It reflects the action [1] of the state machine. In case of an incoming message, the confirmed sequence number TX is updated. After that in all cases a DiscReq message is send and the connection state updated to [sraty_kConnectionClosed](#).

Remarks

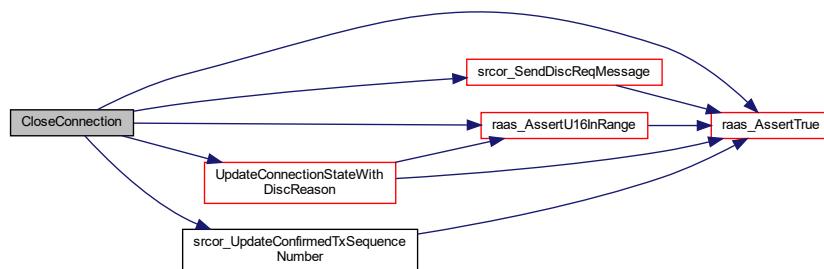
This function shall only be used if the RaSTA connection is already established to the opposite side.

Implements Requirements [RASW-563](#) Process Connection State Machine Function

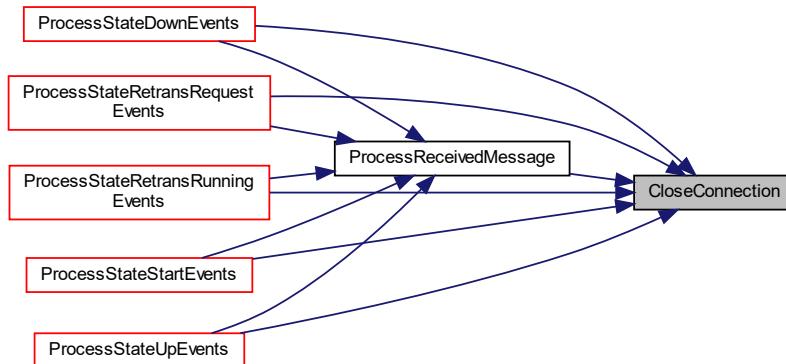
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>disconnect_reason</i>	Disconnect reason in case of a change into the closed state. The reason has a valid range from <code>srtty_kDiscReasonMin</code> $\leq \text{value} < \text{srtty_kDiscReasonMax}$. If the value is outside this range, a <code>radef_kInternalError</code> fatal error is thrown.
in	<i>is_incoming_message</i>	True if triggered by an incoming message.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.2.2 CloseRedundancyChannel() `static void CloseRedundancyChannel (const uint32_t connection_id, const bool is_incoming_message) [static]`

Close the redundancy channel of a specific RaSTA connection.

This internal function is used to close a redundancy channel of a specific connection. It reflects the action [1] of the state machine. In case of an incoming message, the confirmed sequence number TX is updated. Finally the connection state is updated to `srtty_kConnectionClosed`.

Remarks

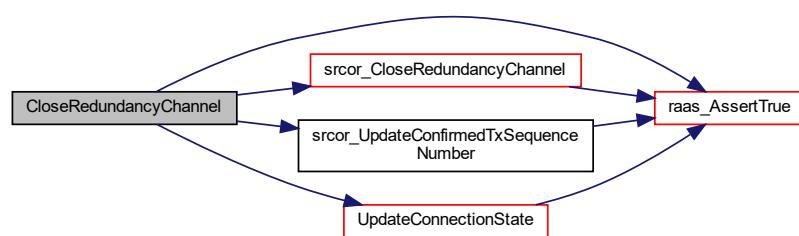
This function shall only be used if the RaSTA connection was not yet established to the opposite side and only the redundancy channels are open.

Implements Requirements RASW-563 Process Connection State Machine Function

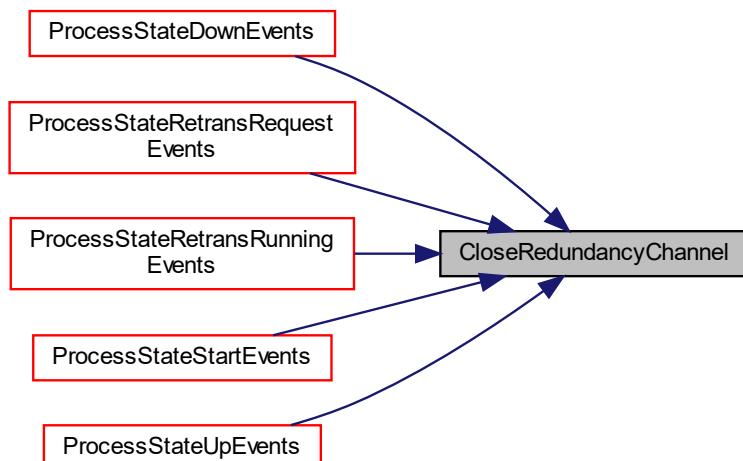
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>is_incoming_message</i>	True if triggered by an incoming message.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.5.2.3 ProcessReceivedMessage() static bool ProcessReceivedMessage (
    const uint32_t connection_id ) [static]
```

Process received message and evaluate timeliness.

This internal function is used to process a received message by calling `srcor_ProcessReceivedMessage`. If the message timeliness is no longer guaranteed, the connection is closed using `CloseConnection` with an `srtiy_kDiscReasonTimeout` as disconnect reason. Finally, the result of the `srcor_ProcessReceivedMessage` is returned.

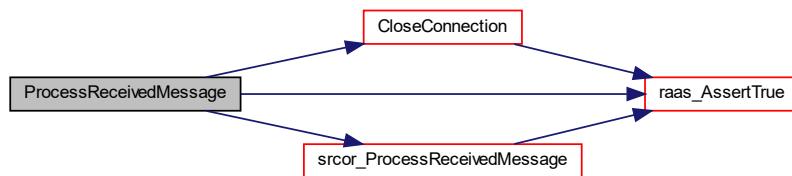
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------------	---

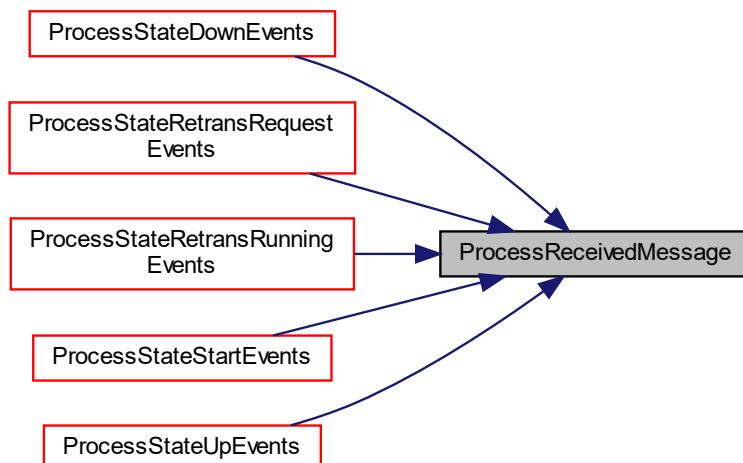
Returns

true, if message timeliness is respected.
false, if message timeliness is no longer guaranteed.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.5.2.4 ProcessStateClosedEvents() static void ProcessStateClosedEvents (
    const uint32_t connection_id,
    const srtyp_ConnectionEvents event ) [static]
```

Process events in state Closed.

This internal function is used to process all incoming events when the state machine is in the "closed"-state. Detailed information about transitions can be found in [srstm_ProcessConnectionStateMachine](#).

Precondition

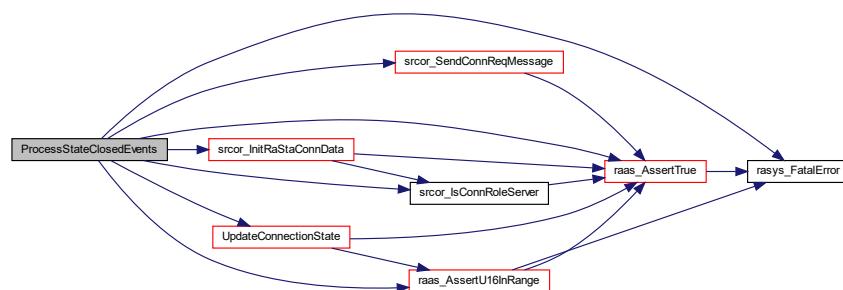
It is checked that the state machine of the given connection is in the [sraty_kConnectionClosed](#) state, otherwise an [radef_kInternalError](#) is thrown.

Implements Requirements [RASW-563](#) Process Connection State Machine Function
[RASW-560](#) sr_state_machine Events

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>event</i>	Received event to process. The event has a valid range from srtyp_kConnEventMin $\leq \text{value} < \text{srtyp_kConnEventMax}$, otherwise a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.5.2.5 ProcessStateDownEvents() static void ProcessStateDownEvents (
    const uint32_t connection_id,
    const srtyp_ConnectionEvents event ) [static]
```

Process events in state Down.

This internal function is used to process all incoming events when the state machine is in the "down"-state. Detailed information about transitions can be found in [srstm_ProcessConnectionStateMachine](#).

Precondition

It is checked that the state machine of the given connection is in the [sraty_kConnectionDown](#) state, otherwise an [radef_kInternalError](#) is thrown.

Implements Requirements [RASW-563](#) Process Connection State Machine Function

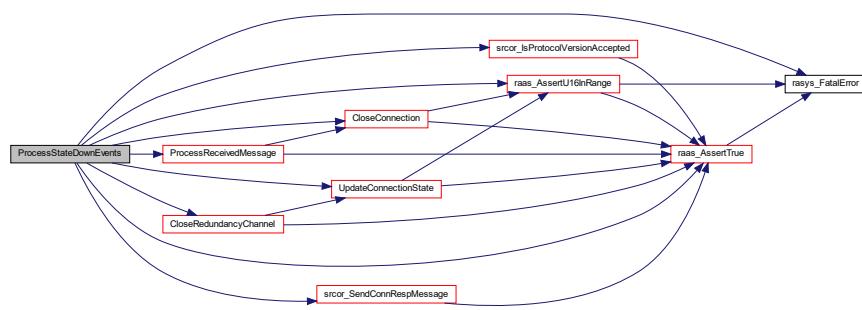
[RASW-560](#) sr_state_machine Events

[RASW-755](#) Process State Machine Receive ConnReq Message Sequence

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>event</i>	Received event to process. The event has a valid range from srtyp_kConnEventMin $\leq \text{value} < \text{srtyp_kConnEventMax}$, otherwise a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.2.6 ProcessStateRetransRequestEvents() static void ProcessStateRetransRequestEvents (const uint32_t connection_id, const srtyp_ConnectionEvents event, const bool sequence_number_in_seq) [static]

Process events in state RetransRequest.

This internal function is used to process all incoming events when the state machine is in the "retransmission request"-state. Detailed information about transitions can be found in [srstm_ProcessConnectionStateMachine](#).

Precondition

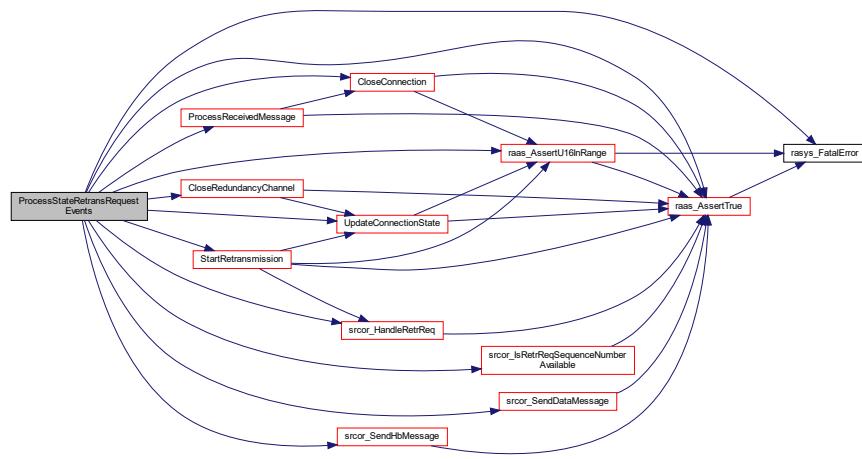
It is checked that the state machine of the given connection is in the [sraty_kConnectionRetransRequest](#) state, otherwise an [radef_kInternalError](#) is thrown.

Implements Requirements [RASW-563](#) Process Connection State Machine Function
[RASW-560](#) sr_state_machine Events

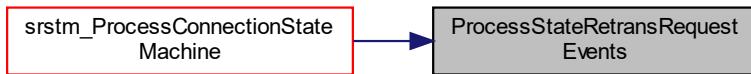
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>event</i>	Received event to process. The event has a valid range from srtyp_kConnEventMin $\leq \text{value} <$ srtyp_kConnEventMax , otherwise a radef_kInternalError fatal error is thrown.
in	<i>sequence_number_in_seq</i>	true, if the received sequence number is in sequence

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.5.2.7 ProcessStateRetransRunningEvents() static void ProcessStateRetransRunningEvents (
    const uint32_t connection_id,
    const srtyp\_ConnectionEvents event,
    const bool sequence_number_in_seq,
    const bool confirmed_time_stamp_in_seq ) [static]
```

Process events in state RetransRunning.

This internal function is used to process all incoming events when the state machine is in the "retransmission running"-state. Detailed information about transitions can be found in [srstm_ProcessConnectionStateMachine](#).

Precondition

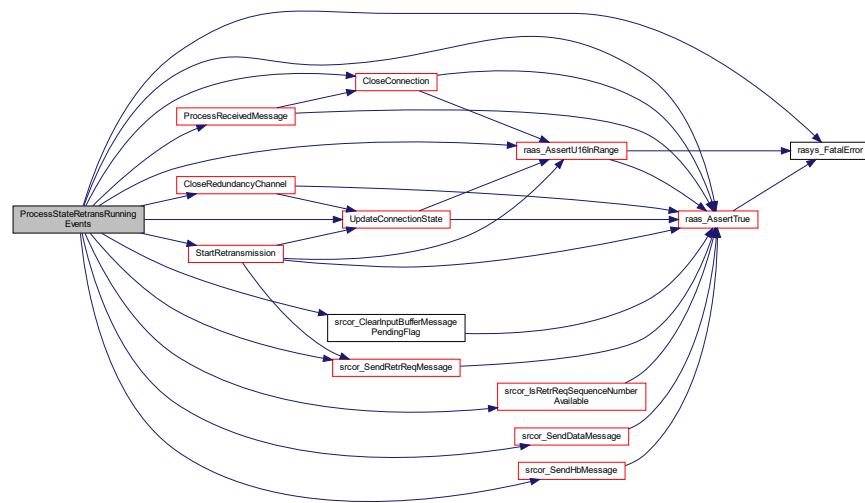
It is checked that the state machine of the given connection is in the `srtyp_kConnectionRetransRunning` state, otherwise an `raef_kInternalError` is thrown.

Implements Requirements [RASW-563](#) Process Connection State Machine Function
[RASW-560](#) sr_state_machine Events

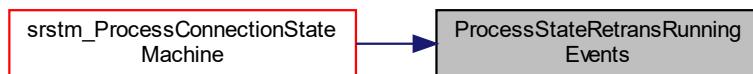
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>event</i>	Received event to process. The event has a valid range from <code>srtyp_kConnEventMin</code> $\leq \text{value} < \text{srtyp_kConnEventMax}$, otherwise a <code>radef_kInternalError</code> fatal error is thrown.
in	<i>sequence_number_in_seq</i>	true, if the received sequence number is in sequence
in	<i>confirmed_time_stamp_in_seq</i>	true, if the confirmed timestamp is in sequence

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.5.2.8 ProcessStateStartEvents() static void ProcessStateStartEvents (
    const uint32_t connection_id,
    const srtyp_ConnectionEvents event,
    const bool sequence_number_in_seq,
    const bool confirmed_time_stamp_in_seq ) [static]
  
```

Process events in state Start.

This internal function is used to process all incoming events when the state machine is in the "start"-state. Detailed information about transitions can be found in [srstm_ProcessConnectionStateMachine](#).

Precondition

It is checked that the state machine of the given connection is in the `srtyp_kConnectionStart` state, otherwise an `raodef_kInternalError` is thrown.

Implements Requirements [RASW-563](#) Process Connection State Machine Function

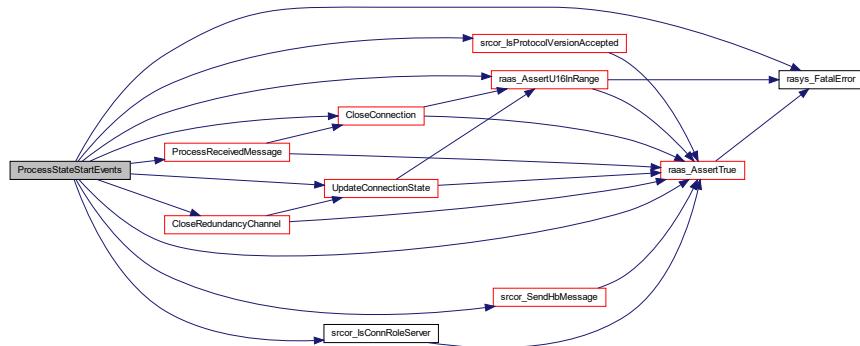
[RASW-560](#) sr_state_machine Events

[RASW-757](#) Process State Machine Receive ConnResp Message Sequence

Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<code>event</code>	Received event to process. The event has a valid range from <code>srtyp_kConnEventMin</code> $\leq \text{value} < \text{srtyp_kConnEventMax}, otherwise a raodef_kInternalError fatal error is thrown.$
in	<code>sequence_number_in_seq</code>	true, if the received sequence number is in sequence
in	<code>confirmed_time_stamp_in_seq</code>	true, if the confirmed timestamp is in sequence

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.5.2.9 ProcessStateUpEvents() static void ProcessStateUpEvents (
```

```
    const uint32_t connection_id,
```

```
    const srtyp_ConnectionEvents event,
```

```
    const bool sequence_number_in_seq,
```

```
    const bool confirmed_time_stamp_in_seq ) [static]
```

Process events in state Up.

This internal function is used to process all incoming events when the state machine is in the "up"-state. Detailed information about transitions can be found in [srstm_ProcessConnectionStateMachine](#).

Precondition

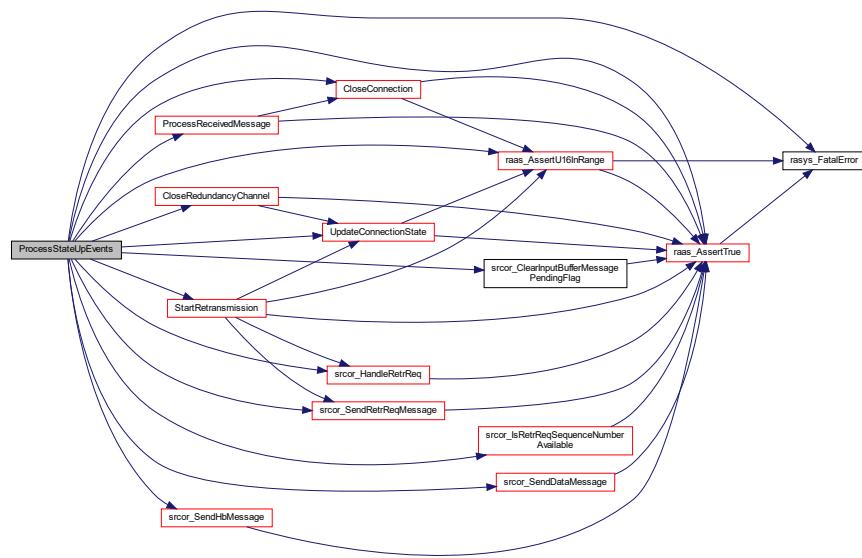
It is checked that the state machine of the given connection is in the [sraty_kConnectionUp](#) state, otherwise an [radef_kInternalError](#) is thrown.

Implements Requirements [RASW-563](#) Process Connection State Machine Function
[RASW-560](#) sr_state_machine Events
[RASW-759](#) Process State Machine Receive Data Message (in Sequence) Sequence
[RASW-761](#) Process State Machine Receive RetrReq Message (in Sequence) Sequence
[RASW-763](#) Process State Machine Receive RetrReq Message (not in Sequence) Sequence

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>event</i>	Received event to process. The event has a valid range from srtyp_kConnEventMin $\leq \text{value} < \text{srtyp_kConnEventMax}, otherwise a radef_kInternalError fatal error is thrown.$
in	<i>sequence_number_in_seq</i>	true, if the received sequence number is in sequence
in	<i>confirmed_time_stamp_in_seq</i>	true, if the confirmed timestamp is in sequence

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.2.10 srstm_GetConnectionState() *srtty_ConnectionStates* srstm_GetConnectionState (const uint32_t connection_id)

Return the state of a RaSTA connection state machine.

This function is used to get the connection state of a dedicated RaSTA connection.

Precondition

The state machine module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-561](#) Get Connection State Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

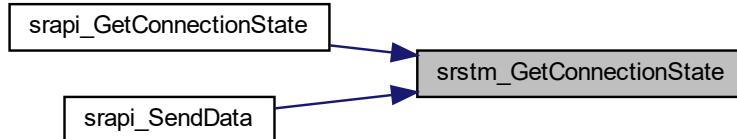
Returns

state of the RaSTA connection state machine

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.2.11 `srstm_Init()` `void srstm_Init (`
`const uint32_t configured_connections)`

Initialize SafRetL state machine module.

This function is used to initialize the state machine module. It saves the passed number of configured connections. A fatal error is raised, if this function is called multiple times.

Precondition

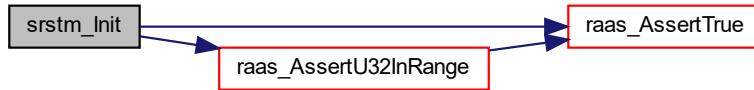
The state machine module must not be initialized, otherwise a [radef_kAlreadyInitialized](#) fatal error is thrown.

Implements Requirements [RASW-562](#) Init sr_state_machine Function

Parameters

in	<code>configured_connections</code>	Number of configured RaSTA connections. Valid range: $1 \leq \text{value} \leq \text{RADEF_MAX_NUMBER_OF_RSTA_CONNECTIONS}$.
----	-------------------------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.2.12 srstm_ProcessConnectionStateMachine() `void srstm_ProcessConnectionStateMachine (`
`const uint32_t connection_id,`
`const srtyp_ConnectionEvents event,`
`const bool sequence_number_in_seq,`
`const bool confirmed_time_stamp_in_seq)`

Process RaSTA connection state machine.

This function is used to process all incoming events of the state machine and launch the needed actions. All details can be found in the image below or in table 18 of the standard "Elektrische Bahn-Signalanlagen – Teil 200: Sicherer Übertragungsprotokoll RaSTA nach DIN EN 50159 (VDE0831-159); Juni 2015":

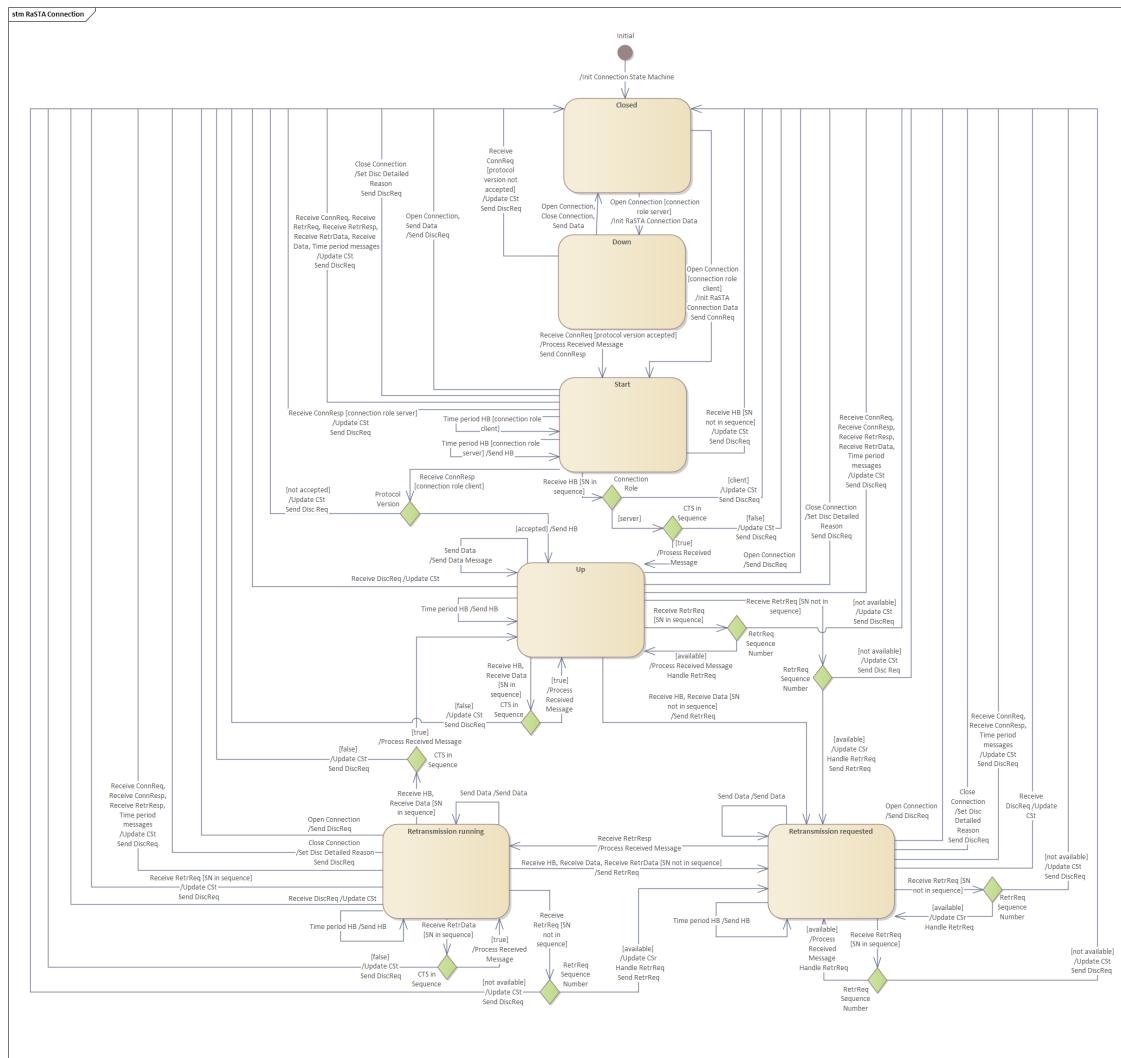


Figure 1 RaSTA SafRetL State Machine

Precondition

The state machine module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements

[RASW-563](#) Process Connection State Machine Function

[RASW-560](#) sr_state_machine Events

[RASW-749](#) Connection State Notification Sequence

[RASW-755](#) Process State Machine Receive ConnReq Message Sequence

[RASW-757](#) Process State Machine Receive ConnResp Message Sequence

[RASW-759](#) Process State Machine Receive Data Message (in Sequence) Sequence

[RASW-761](#) Process State Machine Receive RetrReq Message (in Sequence) Sequence

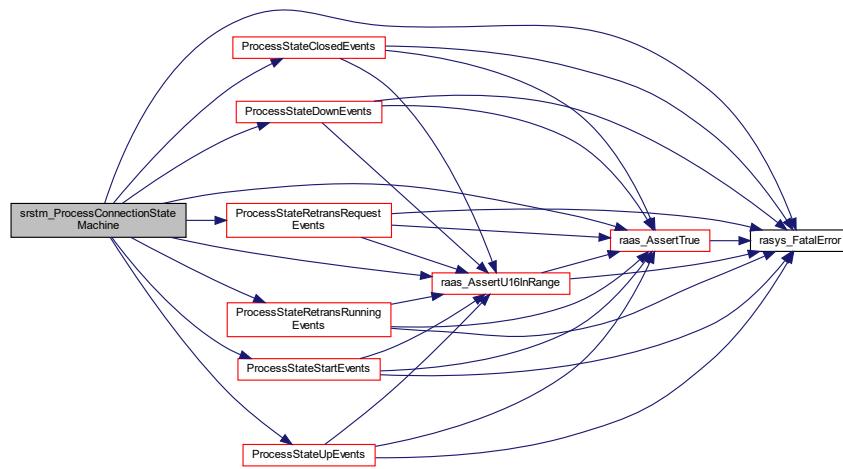
[RASW-763](#) Process State Machine Receive RetrReq Message (not in Sequence) Sequence

[RASW-765](#) Send Pending Messages Sequence

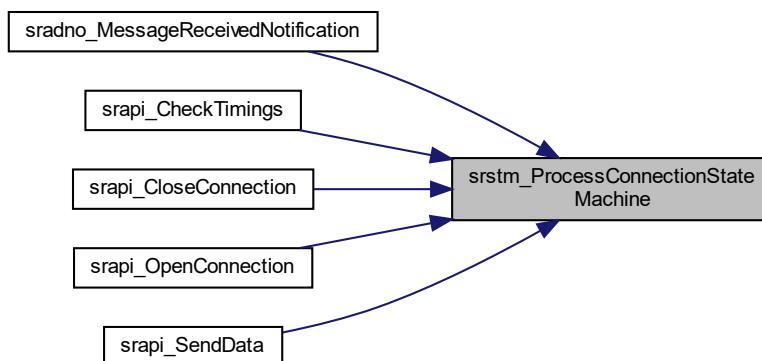
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>event</i>	Received Event to process. The event has a valid range from <code>srtyp_kConnEventMin</code> $\leq \text{value} < \text{srtyp_kConnEventMax}$. If the value is outside this range, a <code>radef_kInvalidParameter</code> fatal error is thrown.
in	<i>sequence_number_in_seq</i>	true, if the received sequence number is in sequence
in	<i>confirmed_time_stamp_in_seq</i>	true, if the confirmed timestamp is in sequence

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.5.2.13 StartRetransmission() static void StartRetransmission (
    const uint32_t connection_id,
    const sraty_ConnectionStates new_state,
    bool retransmission_requested ) [static]
```

Start a retransmission.

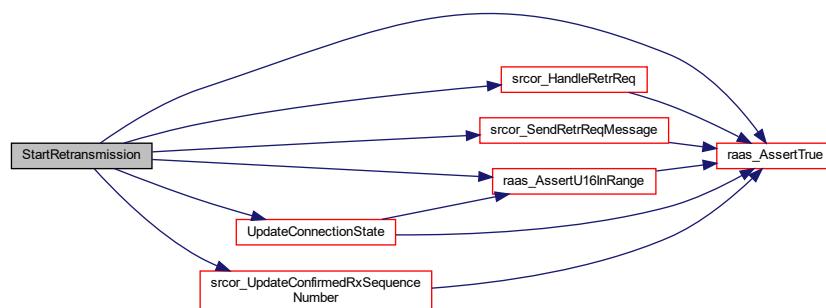
This internal function is used to start a retransmission for a specific connection. It reflects a part of action [4] of the state machine by updating the confirmed sequence number CS_R with the received sequence number CS_PDU. The core module is triggered to handle the retransmission request. If missing messages were detected on this side, a retransmission request is send to the opposite side. Finally the state is updated, if a state change is detected.

Implements Requirements RASW-563 Process Connection State Machine Function

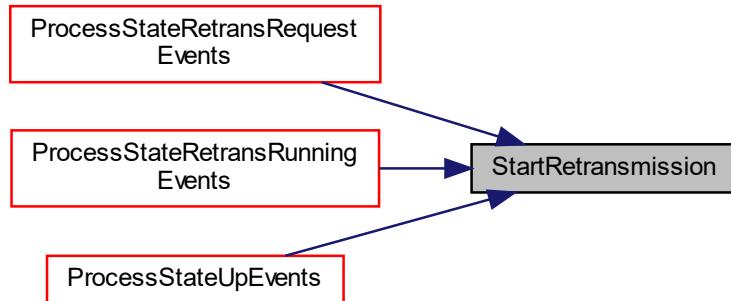
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>new_state</i>	New state to change into. The state has a valid range from <code>sraty_kConnectionMin</code> $\leq \text{value} < \text{sraty_kConnectionMax}$. If the value is outside this range, a <code>radef_kInternalError</code> fatal error is thrown.
in	<i>retransmission_requested</i>	True if a retransmission request must be send to the opposite side

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.2.14 UpdateConnectionState() static void UpdateConnectionState (const uint32_t connection_id, const sraty_ConnectionStates new_state) [static]

Update the connection state and send a connection state notification to the application layer.

This internal function is used to update the connection state of a specific RaSTA connection. Then a connection state notification with the updated data is send to the application.

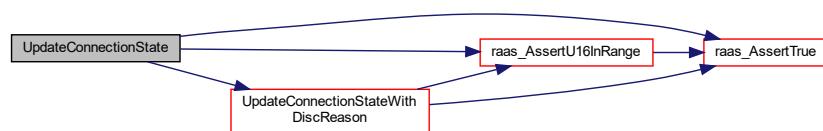
Implements Requirements [RASW-563](#) Process Connection State Machine Function

[RASW-749](#) Connection State Notification Sequence

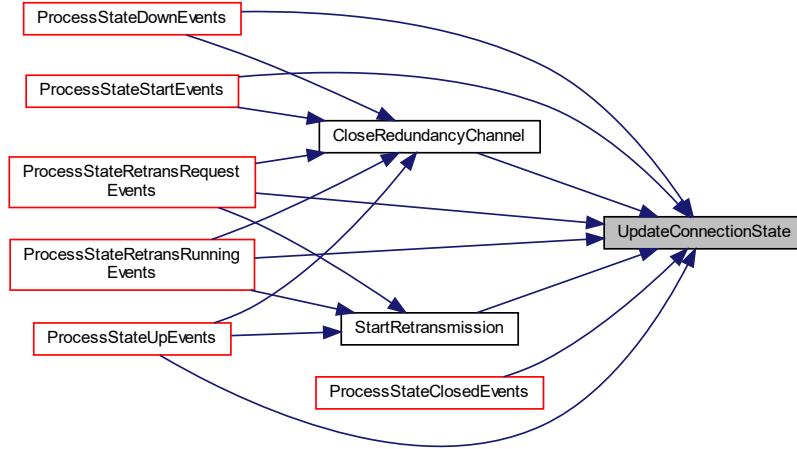
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>new_state</i>	New state to change into. The state has a valid range from sraty_kConnectionMin $\leq \text{value} <$ sraty_kConnectionMax . If the value is outside this range, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.5.2.15 UpdateConnectionStateWithDiscReason() static void UpdateConnectionStateWithDiscReason()
{
    const uint32_t connection_id,
    const sraty_ConnectionStates new_state,
    const sraty_DisconnectReason disconnect_reason ) [static]
```

Update the connection state and send a connection state notification to the application layer.

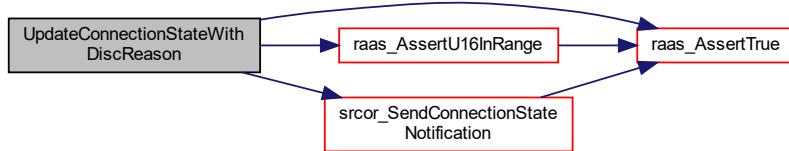
This internal function is used to update the connection state of a specific RaSTA connection. It is used when the connection is closed and a specific disconnection reason shall be passed. Then a connection state notification with the updated data is sent to the application.

Implements Requirements [RASW-563](#) Process Connection State Machine Function
[RASW-749](#) Connection State Notification Sequence

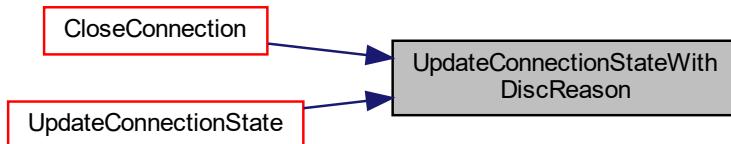
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>new_state</i>	New state to change into. The state has a valid range from <code>sraty_kConnectionMin</code> $\leq \text{value} <$ <code>sraty_kConnectionMax</code> . If the value is outside this range, a <code>radef_kInternalError</code> fatal error is thrown.
in	<i>disconnect_reason</i>	Disconnect reason in case of a change into the closed state. The reason has a valid range from <code>sraty_kDiscReasonMin</code> $\leq \text{value} <$ <code>sraty_kDiscReasonMax</code> . If the value is outside this range, a <code>radef_kInternalError</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6 Core component

Interface of RaSTA SafRetL core module.

Collaboration diagram for Core component:



Classes

- struct `srcor_InputBuffer`
Struct for the newly received message input buffer.
- struct `srcor_TemporaryBuffer`
Struct for the message payload temporary buffer.
- struct `srcor_RaStaConnectionData`
Struct for the process data of a RaSTA connection.

Functions

- static bool `CheckConnectionConfigurations` (const uint32_t number_of_connections, const `srcty_ConnectionConfiguration` *const connection_configurations)

Check if the connection configurations are valid.
- static bool `IsMessageTimeoutRelated` (const `srtyp_SrMessageType` message_type)

Checks if a message is timeout related or not.
- static uint32_t `GetCurrentSequenceNumberAndIncrementNumber` (const uint32_t connection_id)

Get the Current Sequence Number SN_T and Increment Number.
- static bool `SendPendingMessagesWithFlowControlAllowed` (const uint32_t connection_id)

Checks if the flow control allows to send message to the opposite side.
- static void `ReceivedFlowControlCheck` (const uint32_t connection_id, const `srtyp_SrMessageType` message_type)

Perform the received flow control check.
- static bool `GeneralMessageCheck` (const uint32_t connection_id, const `srtyp_SrMessage` *const msg, `srtyp_SrMessageHeader` *const msg_hdr)

Check the general part of a message.
- static bool `CheckSequenceNumberRange` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr)

Check the sequence number range of a new received message.
- static bool `CheckSequenceNumber` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr)

Check the sequence number of a new received message.
- static bool `CheckConfirmedSequenceNumber` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr)

Check the confirmed sequence number of a new received message.
- static bool `CheckTimeStamp` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr)

Checks the time stamp of a new received message.
- static bool `CheckConfirmedTimeStamp` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr)

Checks the confirmed time stamp of a new received message.
- static bool `CalculateTimeliness` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr, const uint32_t current_time)

Calculate the timeliness of a new received message.
- static void `SetConnectionEvent` (const `srtyp_SrMessageType` message_type, `srtyp_ConnectionEvents` *const connection_event)

Set the connection event according to the message type.
- bool `srcor_IsConfigurationValid` (const `srcty_SafetyRetransmissionConfiguration` *const sr_layer_configuration)

Checks if a SafRetL configuration is valid.
- void `srcor_Init` (const `srcty_SafetyRetransmissionConfiguration` *const sr_layer_configuration)

Initialize SafRetL core module.
- `radef_RaStaReturnCode` `srcor_GetConnectionId` (const uint32_t sender_id, const uint32_t receiver_id, uint32_t *const connection_id)

Get the connection identification associated with the specified sender and receiver identification.
- void `srcor_InitRaStaConnData` (const uint32_t connection_id)

Initialize the SafRetL core data of a dedicated RaSTA connection.
- void `srcor_CloseRedundancyChannel` (const uint32_t connection_id)

Close the redundancy channel of a rasta connection and send a diagnostic notification.
- void `srcor_ReceiveMessage` (const uint32_t connection_id, `srtyp_ConnectionEvents` *const connection_event, bool *const sequence_number_in_seq, bool *const confirmed_time_stamp_in_seq)

Read and analyze a received SafRetL message.

- **bool `srcor_ProcessReceivedMessage` (const uint32_t connection_id)**
Process a successfully received message of a dedicated connection.
- **void `srcor_UpdateConfirmedTxSequenceNumber` (const uint32_t connection_id)**
Update confirmed sequence number to transmit from the message in the input buffer.
- **void `srcor_UpdateConfirmedRxSequenceNumber` (const uint32_t connection_id)**
Update confirmed sequence number receive from message in input buffer.
- **bool `srcor_IsProtocolVersionAccepted` (const uint32_t connection_id)**
Checks if the requested protocol version is accepted.
- **void `srcor_SetReceivedMessagePendingFlag` (const uint32_t connection_id)**
Set the message pending flag of a dedicated RaSTA connection.
- **bool `srcor_GetReceivedMessagePendingFlag` (const uint32_t connection_id)**
Get the received message pending flag for a dedicated RaSTA connection.
- **void `srcor_WriteMessagePayloadToTemporaryBuffer` (const uint32_t connection_id, const uint16_t message_payload_size, const uint8_t *const message_payload)**
Write message payload to temporary buffer for messages to send.
- **void `srcor_ClearInputBufferMessagePendingFlag` (const uint32_t connection_id)**
Clear input buffer messages pending flag.
- **void `srcor_SendDataMessage` (const uint32_t connection_id)**
Create and send a SafRetL data message from the temporary buffer.
- **void `srcor_SendConnReqMessage` (const uint32_t connection_id)**
Create and send a SafRetL connection request message.
- **void `srcor_SendConnRespMessage` (const uint32_t connection_id)**
Create and send a SafRetL connection response message.
- **void `srcor_SendDiscReqMessage` (const uint32_t connection_id, const `sraty_DiscReason` disconnect_reason)**
Create and send a SafRetL disconnection request message.
- **void `srcor_SetDiscDetailedReason` (const uint32_t connection_id, const uint16_t detailed_disconnect_reason)**
Set detailed reason for disconnection request message.
- **void `srcor_SendHbMessage` (const uint32_t connection_id)**
Create and send a SafRetL heartbeat message.
- **void `srcor_SendRetrReqMessage` (const uint32_t connection_id)**
Create and send a SafRetL retransmission request message.
- **void `srcor_HandleRetrReq` (const uint32_t connection_id)**
Handle a retransmission request and send a SafRetL retransmission response, heartbeat or data message.
- **bool `srcor_IsRetrReqSequenceNumberAvailable` (const uint32_t connection_id)**
Checks if a requested retransmission sequence number is available in the send buffer.
- **bool `srcor_IsConnRoleServer` (const uint32_t connection_id)**
Returns true, if the own connection role is server.
- **bool `srcor_IsMessageTimeout` (const uint32_t connection_id)**
Checks if a message timeout occurred.
- **bool `srcor_IsHeartbeatInterval` (const uint32_t connection_id)**
Checks if the heartbeat interval is elapsed.
- **bool `srcor_IsReceivedMsgPendingAndBuffersNotFull` (const uint32_t connection_id)**
Check if received messages are pending and send & received buffer are not full.
- **void `srcor_SendPendingMessages` (const uint32_t connection_id)**
Send pending messages from the send buffer, if the flow control allows to send.
- **void `srcor_SendConnectionStateNotification` (const uint32_t connection_id, const `sraty_ConnectionStates` connection_state, const `sraty_DiscReason` disconnect_reason)**
Send a connection state changed notification to the application layer.
- **void `srcor_GetBufferSizeAndUtilisation` (const uint32_t connection_id, `sraty_BufferUtilisation` *const buffer_utilisation, uint16_t *const opposite_buffer_size)**
Get the opposite receive buffer size and the own buffer utilisation for the connection state request.

Variables

- const `srtyp_ProtocolVersion srcor_kProtocolVersion`
Definition of RaSTA protocol version.

4.6.1 Detailed Description

Interface of RaSTA SafRetL core module.

Specification of the Core component of the RaSTA Safety and Retransmission Layer.

This module provides many of the core logic functionality for SafRetL. This contains the following:

- send/receive SafRetL messages
- read messages when available
- handle send & receive buffer
- handle sequence number and confirmed sequence number
- handle protocol version
- handle message timings, heartbeat timings and timeouts
- handle retransmissions

Implements Requirements [RASW-565](#) Component sr_core Overview

[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

[RASW-520](#) Error Handling

[RASW-521](#) Input Parameter Check

4.6.2 Function Documentation

4.6.2.1 CalculateTimeliness() static bool CalculateTimeliness (

```
const uint32_t connection_id,
const srtyp_SrMessageHeader *const msg_hdr,
const uint32_t current_time ) [static]
```

Calculate the timeliness of a new received message.

This internal function calculated the new values for the round trip time (T_{rtd}) and the alive time (T_{alive}). With this updates values, the timer T_i is then set. Following calculation are used:

- $T_{rtd} = T_{local} + T_{granularity} - CTS_PDU$
- $T_{alive} = T_{local} - CTS_R$
- $T_i = T_{max} - T_{rtd}$

If T_{rtd} is greater then T_{max} , the timeliness of new messages can no longer be guaranteed. Timer T_i is set to 0. This function shall only be called for time-out related messages.

Implements Requirements [RASW-580](#) Receive Message Function

[RASW-806](#) Timeliness Check

[RASW-808](#) Timer Ti

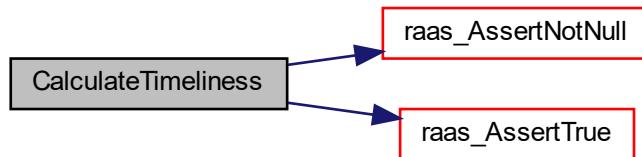
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>msg_hdr</i>	Pointer to message header from new received message. If the pointer is NULL, a radef_klInternalError fatal error is thrown.
in	<i>current_time</i>	Current time used for calculation of T_rtd & T_alive. The full value range is valid and usable.

Returns

true -> timeliness of new messages guaranteed
 false -> timeliness of new messages can no longer be guaranteed

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.2 CheckConfirmedSequenceNumber() static bool CheckConfirmedSequenceNumber (const uint32_t *connection_id*, const [srtyp_SrMessageHeader](#) *const *msg_hdr*) [static]

Check the confirmed sequence number of a new received message.

This internal function checks if the confirmed sequence number of a new received message is in sequence. The following checks are done:

- ConnReq: CS_PDU == 0
- ConnResp: CS_PDU == SN_T -1
- All other message types: $0 \leq \text{CS_PDU} - \text{CS_R} < \text{SN_T} - \text{CS_R}$

If a message doesn't pass this test, it must be discarded.

Implements Requirements [RASW-580](#) Receive Message Function
[RASW-804](#) Sequence Integrity of the Confirmed Sequence Number

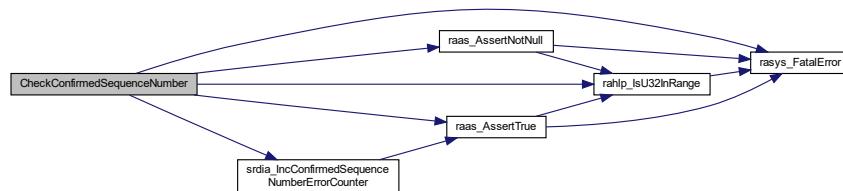
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>msg_hdr</i>	Pointer to message header from new received message. If the pointer is NULL, a radef_KlInternalError fatal error is thrown.

Returns

- true, if confirmed sequence number in sequence.
false, if confirmed sequence number not in sequence.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.6.2.3 CheckConfirmedTimeStamp() static bool CheckConfirmedTimeStamp (
    const uint32_t connection_id,
    const srtyp_SrMessageHeader *const msg_hdr ) [static]
```

Checks the confirmed time stamp of a new received message.

This internal function checks the confirmed time stamp of a new received message. ConnReq, ConnResp, RetrReq, RetrResp and DiscReq are always accepted. For all other message type, the following check is done: $0 \leq \text{CTS_PDU} - \text{CTS_R} < T_{\max}$. The result of this check is returned as is required for processing in the state machine (confirmed time stamp in sequence or not).

Implements Requirements [RASW-580](#) Receive Message Function
[RASW-822](#) Sequence Integrity of the Confirmed Time Stamp

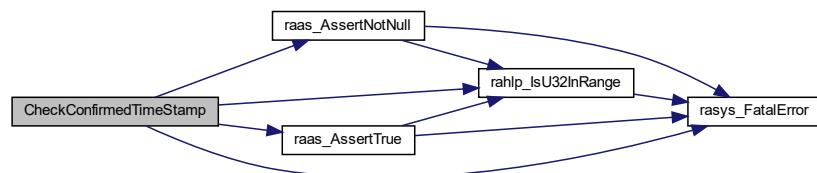
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>msg_hdr</i>	Pointer to message header from new received message. If the pointer is NULL, a radef_KInternalError fatal error is thrown.

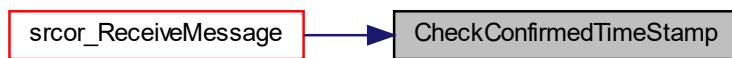
Returns

true, if confirmed time stamp in sequence.
false, if confirmed time stamp not in sequence.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.6.2.4 CheckConnectionConfigurations() static bool CheckConnectionConfigurations (
    const uint32_t number_of_connections,
    const srcty_ConnectionConfiguration *const connection_configurations ) [static]
```

Check if the connection configurations are valid.

This internal function checks if the connection configurations of a SafRetL configuration are valid. All ranges for the SafRetL connection configurations are described in [srcty_SafetyRetransmissionConfiguration::connection_configurations](#). Additionally, it is checked that the sender id is not identical to the receiver id and the connection identification must match the connection index in the array.

Implements Requirements [RASW-573](#) Is Configuration Valid Function

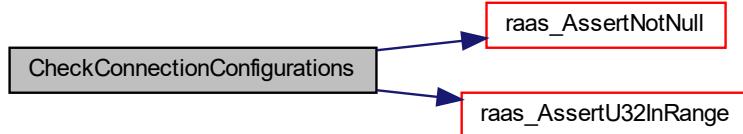
Parameters

in	<i>number_of_connections</i>	Number of configured connections. Valid range of srcty_kMinNumberOfRaStaConnections <= value <= RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS .
in	<i>connection_configurations</i>	Configurations of connections. All ranges for the individual structs are described in srcty_ConnectionConfiguration . Their combination of sender, receiver and connection will be checked by this function for their validity.

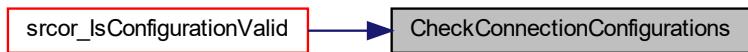
Returns

true, if the connection configuration is valid.
false, if the connection configuration is not valid.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.6.2.5 CheckSequenceNumber() static bool CheckSequenceNumber (
    const uint32_t connection_id,
    const srtyp_SrMessageHeader *const msg_hdr ) [static]
```

Check the sequence number of a new received message.

This internal function checks if the sequence number of a new received message is in sequence. ConnReq, Conn \leftarrow Resp, RetrResp and DiscReq are always accepted. For all other message type, the following check is done: SN_R == SN_PDU. The result of this check is returned as it is required for processing in the state machine (sequence number in sequence or not).

Implements Requirements [RASW-580](#) Receive Message Function
[RASW-805](#) Sequence Number Check

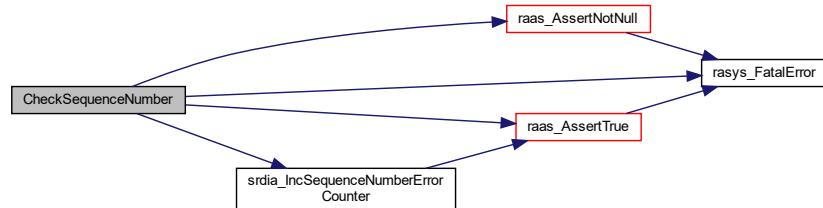
Parameters

in	<i>connection\leftarrow id</i>	RaSTA connection identification. Valid range: $0 \leq$ value < configured number of connections.
in	<i>msg_hdr</i>	Pointer to message header from new received message. If the pointer is NULL, a radef_KInternalError fatal error is thrown.

Returns

true, if sequence number in sequence.
false, if sequence number not in sequence.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.6.2.6 CheckSequenceNumberRange() static bool CheckSequenceNumberRange (
    const uint32_t connection_id,
    const srtyp_SrMessageHeader *const msg_hdr ) [static]
```

Check the sequence number range of a new received message.

This internal function checks the sequence number range of a new received message. ConnReq, ConnResp and RetrResp are always accepted. For all other message type, the following check is done: $0 \leq SN_PDU - SN_R \leq 10 * N_sendMax$. If a message doesn't pass this test, it must be discarded.

Implements Requirements [RASW-580](#) Receive Message Function
[RASW-803](#) Sequence Number Range Check

Parameters

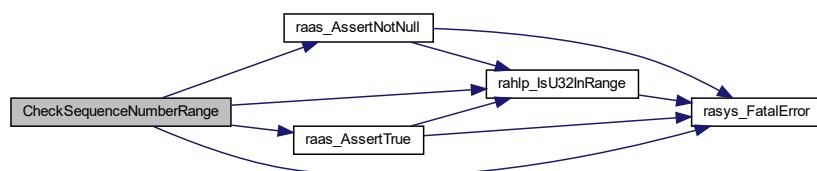
in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>msg_hdr</i>	Pointer to message header from new received message. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Returns

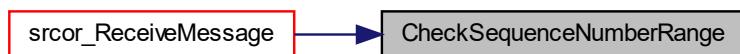
true, if the sequence number range check passed.

false, if the sequence number range check failed.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.6.2.7 CheckTimeStamp() static bool CheckTimeStamp (
    const uint32_t connection_id,
    const srtyp_SrMessageHeader *const msg_hdr ) [static]
```

Checks the time stamp of a new received message.

This internal function checks the time stamp of a new received message. ConnReq, ConnResp, RetrReq, RetrResp and DiscReq are always accepted. For all other message types, the time stamp of a subsequent message (TS_←PDU) cannot be before the time stamp of the previously received message (TS_R). The following check is done: 0 ≤ TS_PDU - TS_R < T_max. If a message doesn't pass this test, it must be discarded.

Implements Requirements RASW-580 Receive Message Function

RASW-806 Timeliness Check

Parameters

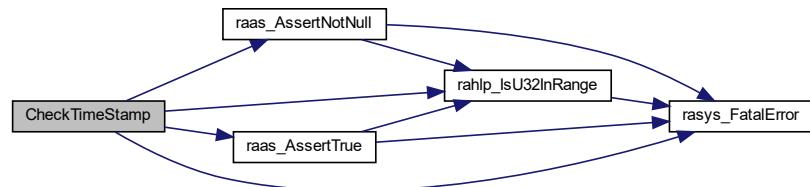
in	<i>connection_↔ id</i>	RaSTA connection identification. Valid range: 0 ≤ value < configured number of connections.
in	<i>msg_hdr</i>	Pointer to message header from new received message. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Returns

true, if time stamp in sequence.

false, if time stamp not in sequence.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.6.2.8 GeneralMessageCheck() static bool GeneralMessageCheck (
    const uint32_t connection_id,
    const srtyp_SrMessage *const msg,
    srtyp_SrMessageHeader *const msg_hdr ) [static]
```

Check the general part of a message.

This internal functions checks the general part of a message. This contains the MD4, message length, message type and is checked with `srmsg_CheckMessage`. If all checks pass, the message header is extracted using `srmsg_GetMessageHeader` and returned via the output parameter `msg_hdr`. Additionally the authenticity of the message is then checked. If a check fails, the corresponding error counter is incremented (`srdia_IncSafetyCodeErrorCounter`, `srdia_IncTypeErrorHandler` & `srdia_IncAddressErrorHandler`) and false is returned to indicate that a test failed.

Implements Requirements RASW-580 Receive Message Function

Parameters

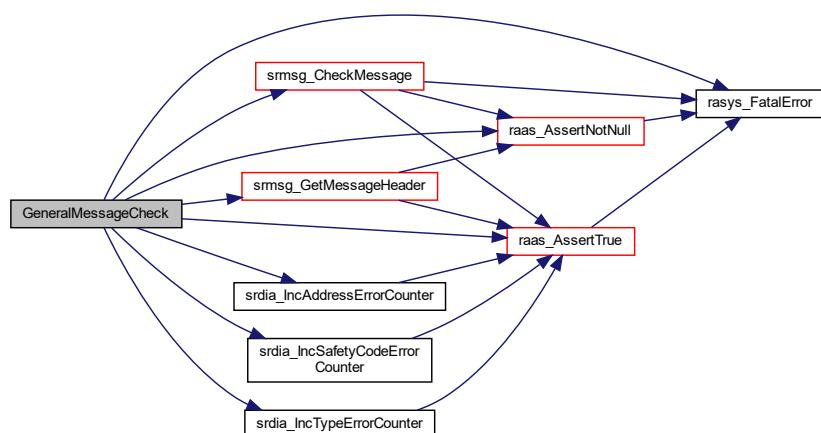
in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<code>msg</code>	Pointer to a memory block containing a message. If the pointer is NULL, a <code>radef_kInternalError</code> fatal error is thrown.
out	<code>msg_hdr</code>	Pointer to a memory block for the message header. If the pointer is NULL, a <code>radef_kInternalError</code> fatal error is thrown.

Returns

true, if all checks passed.

false, if one of the checks fails.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.9 GetCurrentSequenceNumberAndIncrementNumber() static uint32_t GetCurrentSequenceNumberAndIncrementNumber (const uint32_t connection_id) [static]

Get the Current Sequence Number SN_T and Increment Number.

This internal function returns the current sequence number tx (SN_T) of a specific RaSTA connection and increments the sequence number counter afterwards.

Implements Requirements

- [RASW-584](#) Send Data Message Function
- [RASW-582](#) Send ConnReq Message Function
- [RASW-583](#) Send ConnResp Message Function
- [RASW-585](#) Send DiscReq Message Function
- [RASW-586](#) Send Heartbeat Message Function
- [RASW-588](#) Send RetrReq Message Function
- [RASW-570](#) Handle Retransmission Request Function

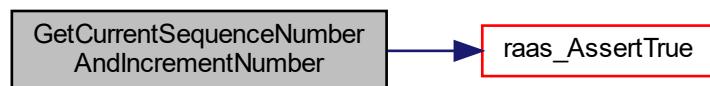
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

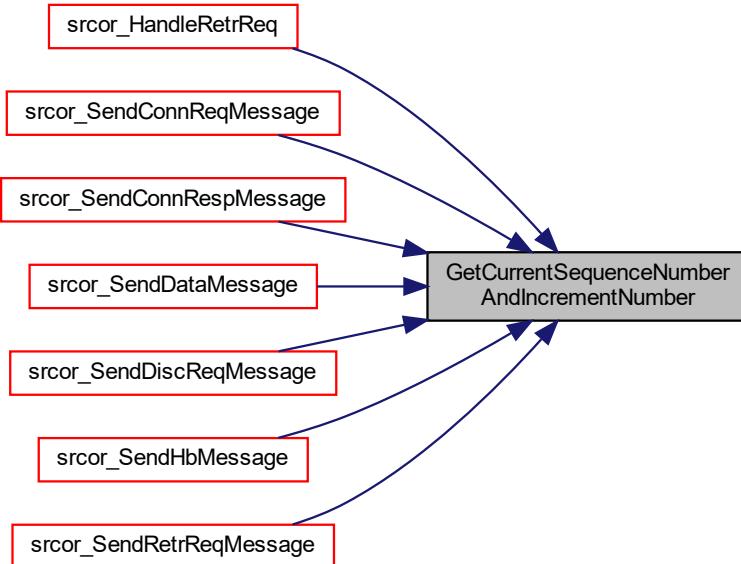
Returns

uint32_t Current sequence number of the passed RaSTA Connection

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.10 IsMessageTimeoutRelated() static bool IsMessageTimeoutRelated (const `srtyp_SrMessageType` `message_type`) [static]

Checks if a message is timeout related or not.

This internal functions checks if a message is timeout related or not. This is checked with its message type. Timeout related message types are:

- heartbeat
- data
- retransmitted data

Implements Requirements [RASW-579](#) Process Received Messages Function

Parameters

in	<code>message_type</code>	Message type to check. Valid range: <code>srtyp_kSrMessageMin</code> <= value <= <code>srtyp_kSrMessageMax</code> - 1.
----	---------------------------	--

Returns

true, if message is timeout related.
false, if message is not timeout related.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.11 ReceivedFlowControlCheck() static void ReceivedFlowControlCheck (const uint32_t connection_id, const srtyp_SrMessageType message_type) [static]

Perform the received flow control check.

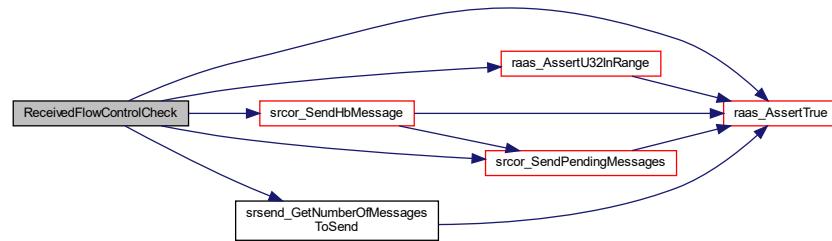
This internal function checks and performs the received flow control. The check is only performed for HB, RetrResp, Data & RetrData messages, as for this message types no answer from the opposite side is assumed. For other message types nothing is done. If the not confirmed sequence numbers are bigger then the MWA from the configuration, a message must be sent to the opposite side to confirm the sequence number. If messages are available in the send buffer, they are send using the [srcor_SendPendingMessages](#) function, otherwise a HB message is send with [srcor_SendHbMessage](#). The not confirmed sequence number is the difference between the last received sequence number to be confirmed (CS_T) and the last effective send confirmed sequence number ([srcor_RaStaConnectionData::confirmed_sequence_number_tx - srcor_RaStaConnectionData::last_send_confirmed_sequence_number_tx](#)).

Implements Requirements [RASW-579](#) Process Received Messages Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>message_type</i>	Message type of the processed message received before the check. Valid range: srtyp_kSrMessageMin $\leq \text{value} \leq \text{srtyp_kSrMessageMax} - 1$.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.12 SendPendingMessagesWithFlowControlAllowed() static bool SendPendingMessagesWithFlowControlAllowed (const uint32_t connection_id) [static]

Checks if the flow control allows to send message to the opposite side.

This internal function checks if there are messages in the send buffer that wait to be transmitted and if the flow control allows new messages to be send. For the flow control to allow sending, the number of unconfirmed send messages in the send buffer should be smaller then the opposite send buffer (`N_sendMax`). Formula: not confirmed messages in send buffer < opposite `N_sendMax`. This parameter is exchanged during connection establishment and saved in the internal structure.

Implements Requirements RASW-587 Send Pending Messages Function

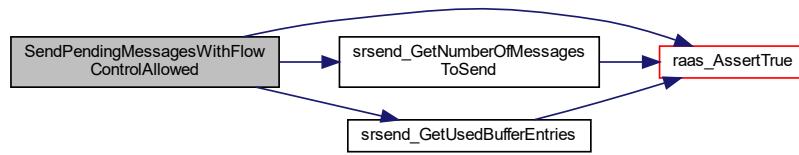
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
----	----------------------------	---

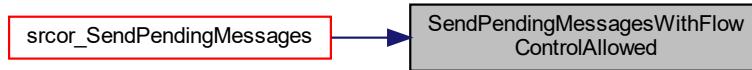
Returns

true, if the flow control allow sending messages.
 false, if the flow control prevents sending messages.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.13 SetConnectionEvent() static void SetConnectionEvent (const [srtp_SrMessageType](#) message_type, [srtp_ConnectionEvents](#) *const connection_event) [static]

Set the connection event according to the message type.

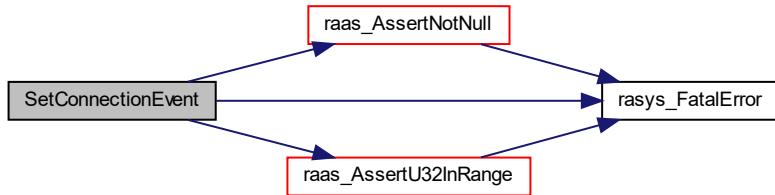
This internal functions set the [srtp_ConnectionEvents](#) for a new received message according to the passed [srtp_SrMessageType](#). Always the matching connection event is set for every message type. This connection event is used to pass the state machine the matching event for a received message.

Implements Requirements [RASW-580](#) Receive Message Function

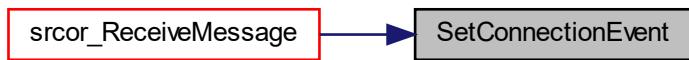
Parameters

in	<i>message_type</i>	Message type. Valid range: srtp_kSrMessageMin <= value < srtp_kSrMessageMax . Only defined values of the message type are accepted.
out	<i>connection_event</i>	Pointer to the connection event. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.14 `srcor_ClearInputBufferMessagePendingFlag()` `void srcor_ClearInputBufferMessagePendingFlag(`
`const uint32_t connection_id)`

Clear input buffer messages pending flag.

This function is used to clear the message pending flag in the input buffer.

Precondition

The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-566](#) Clear Input Buffer Message Pending Flag Function

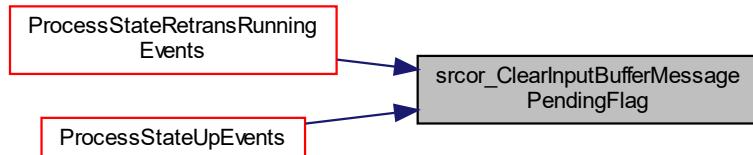
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
----	----------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.15 srcor_CloseRedundancyChannel() `void srcor_CloseRedundancyChannel (const uint32_t connection_id)`

Close the redundancy channel of a rasta connection and send a diagnostic notification.

This function closes the matching redundancy channel to the passed connection (1:1 mapping between the connection id and redundancy channel id). Finally a diagnostic notification is send to the application layer.

Precondition

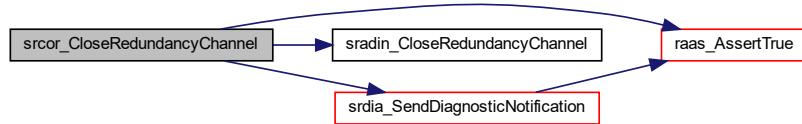
The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-826](#) Close Redundancy Channel Function

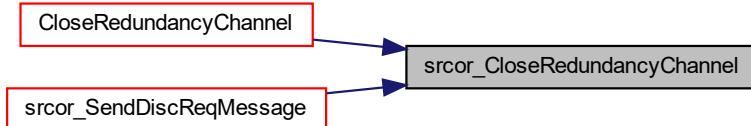
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.16 `srcor_GetBufferSizeAndUtilisation()` `void srcor_GetBufferSizeAndUtilisation (`
`const uint32_t connection_id,`
`sraty_BufferUtilisation *const buffer_utilisation,`
`uint16_t *const opposite_buffer_size)`

Get the opposite receive buffer size and the own buffer utilisation for the connection state request.

This function gets the utilisation information of the send and receive buffer as well as the size of the opposite input buffer (If opposite buffer is 0, connection establishment was not yet executed during which the input buffer size is exchanged).

Precondition

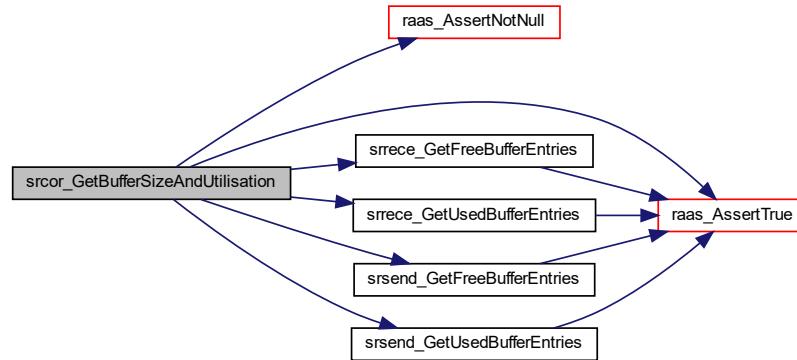
The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-567](#) Get Buffer Size and Utilisation Function

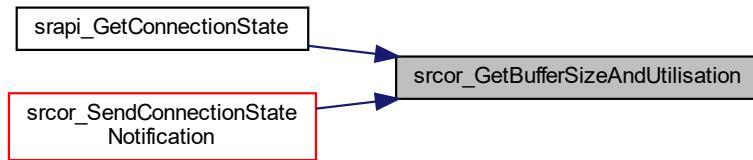
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
out	<code>buffer_utilisation</code>	Pointer to buffer utilization structure. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.
out	<code>opposite_buffer_size</code>	Pointer to pass size of the receive buffer of the connection party. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.17 srcor_GetConnectionId() `radef_RaStaReturnCode` srcor_GetConnectionId (
 const uint32_t sender_id,
 const uint32_t receiver_id,
 uint32_t *const connection_id)

Get the connection identification associated with the specified sender and receiver identification.

This function searches in the saved configuration for a matching connection id to the passed pair of sender and receiver id. If a matching entry is found, the connection id is returned via the out parameter. Otherwise a `radef_kInvalidParameter` is returned.

Precondition

The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements RASW-568 Get Connection ID Function

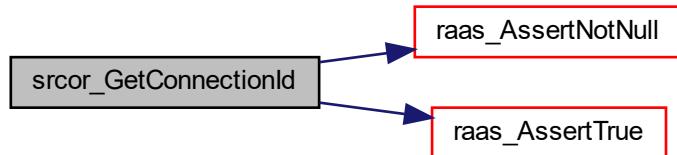
Parameters

in	<i>sender_id</i>	Sender identification. The full value range is valid and usable.
in	<i>receiver_id</i>	Receiver identification. The full value range is valid and usable.
out	<i>connection_id</i>	Pointer to RaSTA connection identification. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Returns

[radef_kNoError](#) if a matching connection ID is in the configuration
[radef_kInvalidParameter](#) if no matching connection ID is in the configuration

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.18 srcor_GetReceivedMessagePendingFlag() `bool srcor_GetReceivedMessagePendingFlag (const uint32_t connection_id)`

Get the received message pending flag for a dedicated RaSTA connection.

This function returns the received message pending flag of a dedicated connection.

Precondition

The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-569](#) Get Received Message Pending Flag Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

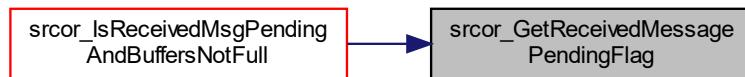
Returns

true, if a received message is pending.
false, if no received message is pending.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.19 srcor_HandleRetrReq() void srcor_HandleRetrReq (const uint32_t *connection_id*)

Handle a retransmission request and send a SafRetL retransmission response, heartbeat or data message.

This function get the current sequence number tx (SN_T) and prepares a message header for new messages. This is then passed to the [srsend_PreparesBufferForRetr](#) function to prepare the send buffer for a retransmission. If the preparation of the send buffer succeeded, the final sequence number is saved back in the local variable. Otherwise a [radef_kInvalidSequenceNumber](#) fatal error is thrown.

Precondition

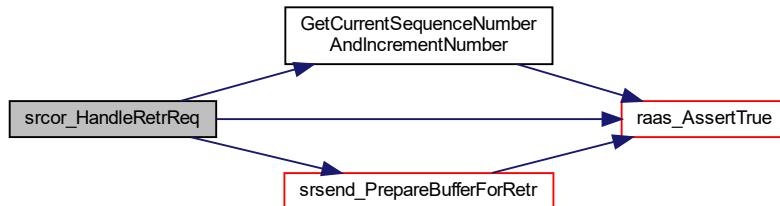
The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-570](#) Handle Retransmission Request Function
[RASW-777](#) Handle Retransmission Request Sequence

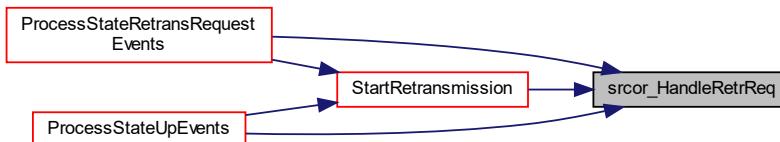
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.20 `srcor_Init()` `void srcor_Init (const srcty_SafetyRetransmissionConfiguration *const sr_layer_configuration)`

Initialize SafRetL core module.

This function is used to initialize the core module. After checking the configuration for validity, the pointer to the configuration is saved internally. If it is not valid, a `radef_kInvalidConfiguration` fatal error is thrown. The configuration is then used to initialize all modules used by the core module:

- messages
- diagnostics
- send buffer
- receive buffer

Finally, the internal structures are also properly initialized.

Precondition

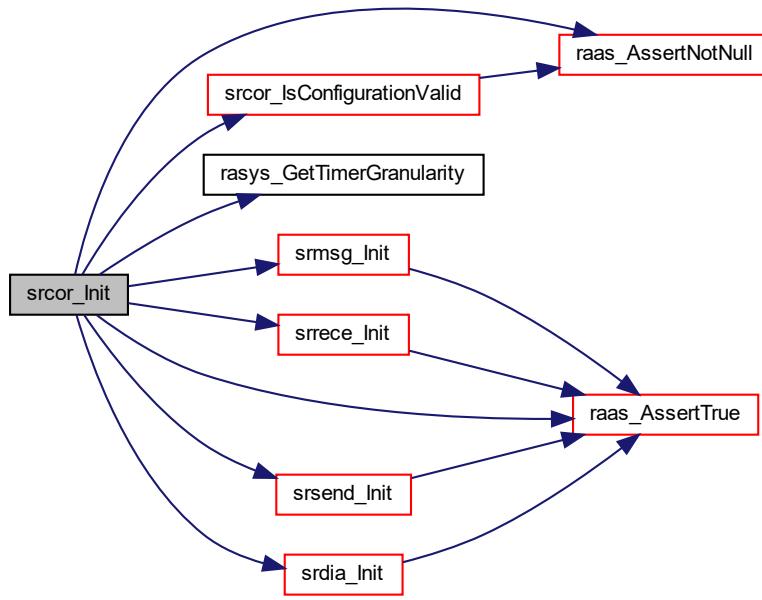
The core module must not be initialized, otherwise a `radef_kAlreadyInitialized` fatal error is thrown.

Implements Requirements [RASW-571](#) Init sr_core Function

Parameters

in	<i>sr_layer_configuration</i>	Pointer to SafRetL configuration. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
----	-------------------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.21 srcor_InitRaStaConnData() `void srcor_InitRaStaConnData (const uint32_t connection_id)`

Initialize the SafRetL core data of a dedicated RaSTA connection.

This function is used to initialize the core data of a specific connection. This means it initializes SN_T with a random number. In case of client, CS_T is set to 0 and CTS_R is set to the current time. Additionally, the corresponding diagnostic module, send and receive buffer are initialized for this connection and the redundancy channel opened.

Precondition

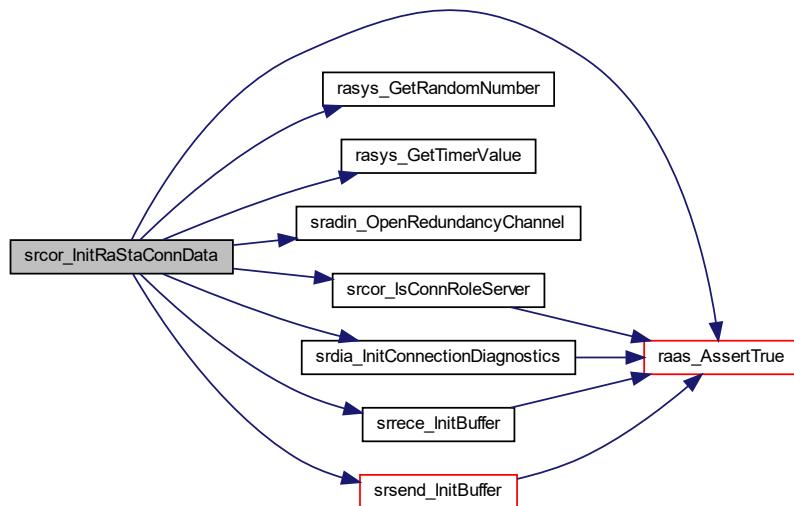
The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-572](#) Init RaSTA Connection Data Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.6.2.22 srcor_IsConfigurationValid() bool srcor_IsConfigurationValid (
    const srcty_SafetyRetransmissionConfiguration *const sr_layer_configuration )

```

Checks if a SafRetL configuration is valid.

This function is used to check if a SafRetL configuration is valid or not. Therefore the configuration is checked if all parameter are within the defined ranges. All ranges for the SafRetL configuration are described in [srcty_SafetyRetransmissionConfiguration](#). Additionally, a few checks on the configuration are done:

- received flow control: MWA < NsendMax
- sender id != receiver id

Implements Requirements [RASW-573](#) Is Configuration Valid Function

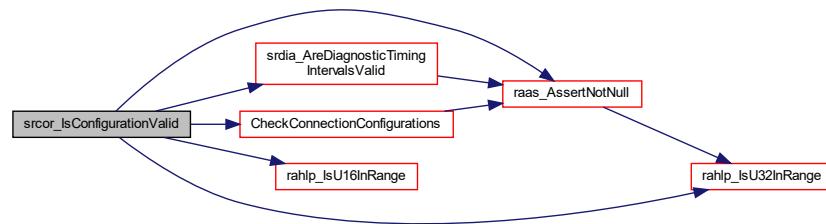
Parameters

in	<i>sr_layer_configuration</i>	Pointer to SafRetL configuration. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
----	-------------------------------	--

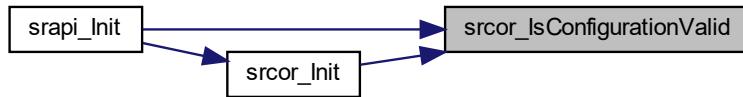
Returns

- true, if the configuration is valid.
false, if the configuration is invalid.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.23 scor_IsConnRoleServer() `bool scor_IsConnRoleServer (const uint32_t connection_id)`

Returns true, if the own connection role is server.

This function checks if the own connection role is server or client. The communication partner with the higher value is server, the one with the lower value client.

Precondition

The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements `RASW-574` Is Connection Role Server Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Returns

true, if the connection role is server.

false, if the connection role is client.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.24 srcor_IsHeartbeatInterval() `bool srcor_IsHeartbeatInterval (const uint32_t connection_id)`

Checks if the heartbeat interval is elapsed.

This function checks if a heartbeat timeout for a dedicated RaSTA connection occurred. This is the case, if the current time - last send timestamp is bigger than the heartbeat period. Formula: current time - last send timestamp $> T_h$.

Precondition

The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-575](#) Is Heartbeat Interval Function

[RASW-807](#) Timer Th

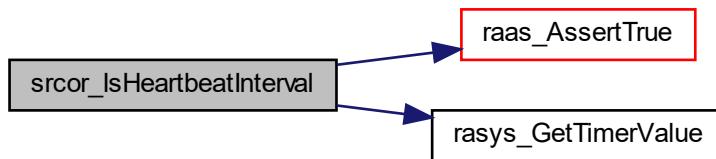
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------------	---

Returns

- true, if the heartbeat interval is elapsed.
- false, if the heartbeat interval is not elapsed.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.25 `srcor_IsMessageTimeout()` `bool srcor_IsMessageTimeout (const uint32_t connection_id)`

Checks if a message timeout occurred.

This function checks if a message timeout for a dedicated RaSTA connection occurred. This is the case, if the current time - last received time-out related timestamp is bigger than maximum accepted age of a message. Formula: $\text{current time} - \text{CTS_R} > T_i$ (T_i is dynamically calculated when receiving a timeout related message $T_i = t_{\max} - t_{\text{rtd}}$)

Precondition

The core module must be initialized, otherwise a `raodef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-576](#) Is Message Timeout Function

[RASW-808](#) Timer Ti

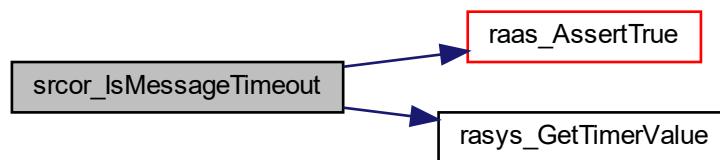
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

Returns

true, if a message timeout occurred.
false, if no message timeout occurred.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.26 `srcor_IsProtocolVersionAccepted()` `bool srcor_IsProtocolVersionAccepted (const uint32_t connection_id)`

Checks if the requested protocol version is accepted.

Checks if the protocol version from a received ConnReq or ConnResp message in the input buffer is on the same or higher version than itself (see RASW-109 for defined version). If the version is smaller, the connection is rejected.

Precondition

The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-577](#) Is Protocol Version Accepted Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

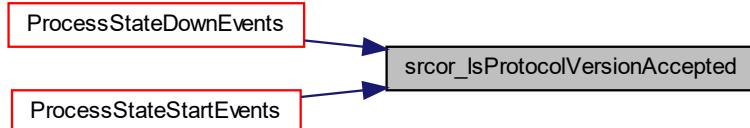
Returns

true, if the protocol version is accepted.
false, if the protocol version is not accepted.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.27 srcor_IsReceivedMsgPendingAndBuffersNotFull() `bool srcor_IsReceivedMsgPendingAndBuffersNotFull (const uint32_t connection_id)`

Check if received messages are pending and send & received buffer are not full.

This function checks if:

- received messages pending
- at least three free entries in send buffer (in worst case of a retransmission of a retransmission, this adds a RetrResp, HB & RetrReq to the send buffer to the already existing data)
- at least one free entry in received buffer

Returns true if all 3 conditions are true.

Implements Requirements [RASW-830](#) Is Received Message Pending And Buffers Not Full Function

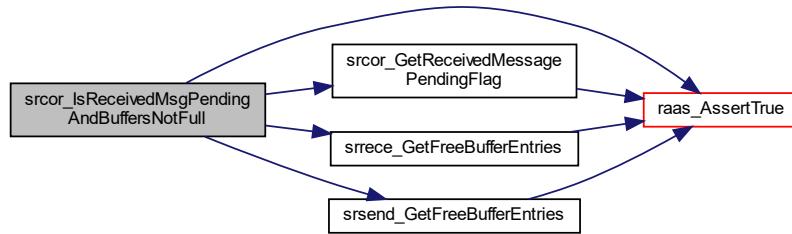
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

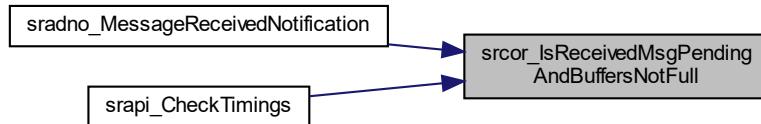
Returns

true -> message pending and free space in buffers
 false -> one or more of the 3 conditions are not given anymore

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.28 `srcor_IsRetrReqSequenceNumberAvailable()` `bool srcor_IsRetrReqSequenceNumberAvailable(const uint32_t connection_id)`

Checks if a requested retransmission sequence number is available in the send buffer.

This function checks if the following sequence number of the last received confirmed sequence number (taken from the saved RetrReq message in the input buffer) is available for retransmission in the send buffer.

Precondition

The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-578](#) Is Retransmission Request Sequence Number Available Function

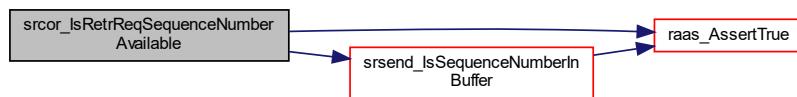
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------------	---

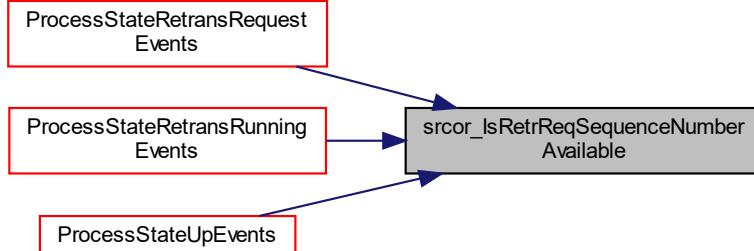
Returns

true, if the requested sequence number is available.
false, if the requested sequence number is not available or no message in input buffer.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.29 srcor_ProcessReceivedMessage() `bool srcor_ProcessReceivedMessage (const uint32_t connection_id)`

Process a successfully received message of a dedicated connection.

This function processes a received message that was copied before in the input buffer of the core module. First it is checked if a message is in the input buffer. If not, a [raodef_kInternalError](#) is thrown. Then the following checks are performed:

- message header is extracted from the message in the input buffer
- current time is obtained with [rasys_GetTimerValue](#)

- for time-out relevant messages ([IsMessageTimeoutRelated](#)), timeliness is checked and timer T_i updated with [CalculateTimeliness](#)

All following checks are only performed if the message timeliness is still guaranteed (timer T_i > 0).

- update local sequence numbers and timestamps
 - for all message types: SN_R = SN_PDU+1, CS_T = SN_PDU, CS_R = CS_PDU, TS_R = TS_PDU
 - for timeout relevant messages ([IsMessageTimeoutRelated](#)): CTS_R = CTS_PDU
 - for ConnReq additionally set: CS_R = SN_T-1, CTS_R = current time
- remove confirmed messages from the send buffer ([srsend_RemoveFromBuffer](#)), if the confirmed sequence number rx (CS_R) changed and the message is not a [srtyp_kSrMessageConnReq](#).
- for ConnReq/ConnResp, save the opposite receive buffer size (N_sendMax)
- add Data/RetrData message payload to received buffer ([srrece_AddToBuffer](#))
- apply received flow control for timeout related messages ([ReceivedFlowControlCheck](#))
- finally for time-out relevant messages ([IsMessageTimeoutRelated](#)), update connection diagnostics ([srdia_UpdateConnectionDiagnostics](#))

In all cases, the message in input buffer flag is cleared.

Precondition

The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-579](#) Process Received Messages Function

[RASW-775](#) Process Received Message Sequence

Parameters

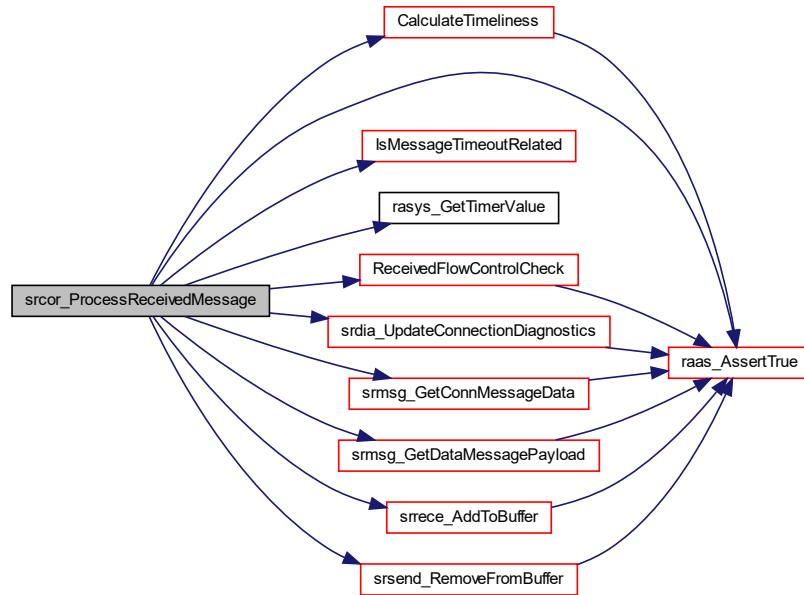
in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

Returns

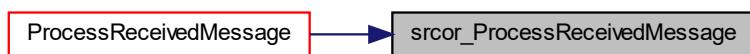
true, if message timeliness is respected (Timer $T_i > 0$).

false, if message timeliness is no longer guaranteed (Timer $T_i == 0$). The caller must close the connection.

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.6.2.30 srcor_ReceiveMessage() void srcor_ReceiveMessage (
    const uint32_t connection_id,
    srtyp_ConnectionEvents *const connection_event,
    bool *const sequence_number_in_seq,
    bool *const confirmed_time_stamp_in_seq )

```

Read and analyze a received SafRetL message.

This function receives and checks a message of a dedicated connection. First, a message is read from the SafRetL adapter interface (`sradin_ReadMessage`). If `radef_kNoMessageReceived` is returned, the `received_data_pending` flag is reset and the function finishes. Otherwise a set of checks is now started to validate the received message:

- general checks are done ([GeneralMessageCheck](#)), stop if check fails
 - MD4
 - message type
 - message size
 - authenticity of the message
- check sequence number range ([CheckSequenceNumberRange](#)), stop if check fails
- check confirmed sequence number ([CheckConfirmedSequenceNumber](#)), stop if check fails
- check if sequence number is in sequence ([CheckSequenceNumber](#)) and save result for state machine (sequence number in sequence)
- check if confirmed time stamp is in sequence ([CheckConfirmedTimeStamp](#)) and save result for state machine (confirmed time stamp in sequence)
- in case of error for safety code, msg type, address, sequence number or confirmed sequence number checks, the corresponding error counter in the diagnostics module is incremented
- set the matching connection event according to message type ([SetConnectionEvent](#))
- if all necessary checks pass, set flag that message is in input buffer

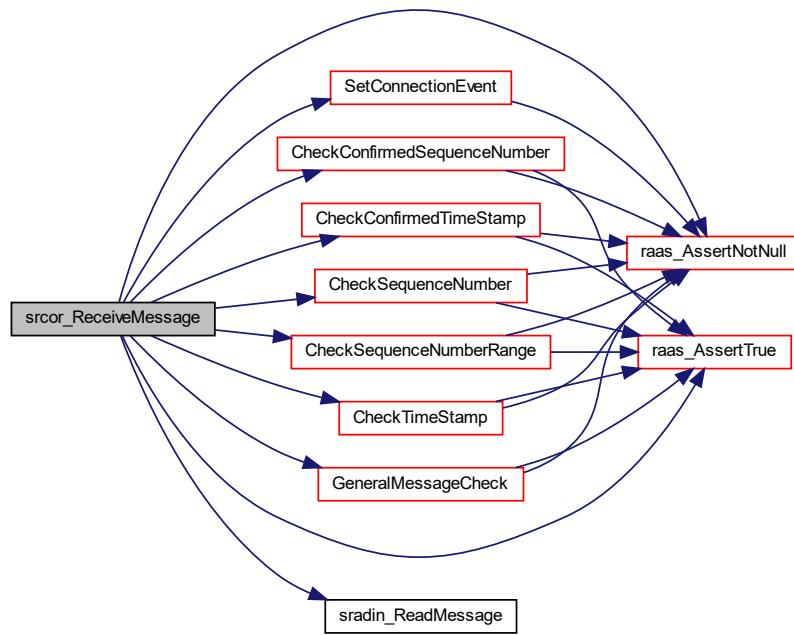
Implements Requirements [RASW-580](#) Receive Message Function

[RASW-771](#) Receive Message Sequence

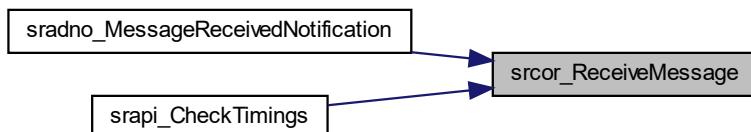
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
out	<i>connection_event</i>	RaSTA connection event to be processed by the connection state machine (srtyp_kConnEventNone is returned, if there is nothing to do for the state machine). If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
out	<i>sequence_number_in_seq</i>	True, if the received sequence number is in sequence. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
out	<i>confirmed_time_stamp_in_seq</i>	True, if the confirmed timestamp is in sequence. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.31 `srcor_SendConnectionStateNotification()` `void srcor_SendConnectionStateNotification (`
`const uint32_t connection_id,`
`const sraty_ConnectionStates connection_state,`
`const sraty_DisconnectReason disconnect_reason)`

Send a connection state changed notification to the application layer.

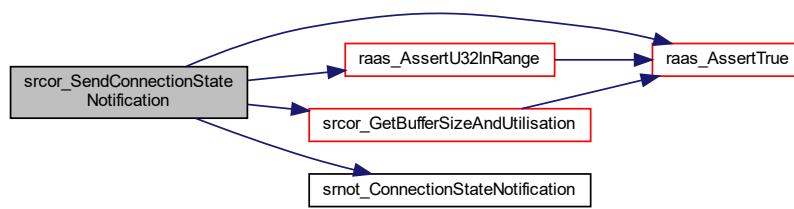
This function collects all necessary data for a connection notification and send this to the application layer. This data includes the utilisation of send and receive buffer, opposite buffer size, connection state and detailed disconnection reason. Detailed disconnection reason is only valid if the connection state is `sraty_kConnectionClosed`.

Implements Requirements [RASW-581](#) Send Connection State Notification Function

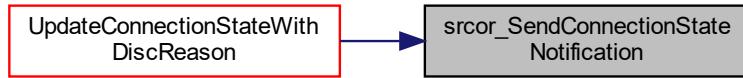
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>connection_state</i>	Connection state. Valid range: <code>sraty_kConnectionMin</code> $\leq \text{value} < \text{sraty_kConnectionMax}$.
in	<i>disconnect_reason</i>	Disconnect reason (only valid in disconnected state). Valid range: <code>sraty_kDiscReasonMin</code> $\leq \text{value} < \text{sraty_kDiscReasonMax}$.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.32 srcor_SendConnReqMessage() `void srcor_SendConnReqMessage (const uint32_t connection_id)`

Create and send a SafRetL connection request message.

This function prepares a new message header (TS_R must be set to 0) that is used to create a new connection request message. This message is then added to the send buffer and send with [srcor_SendPendingMessages](#).

Precondition

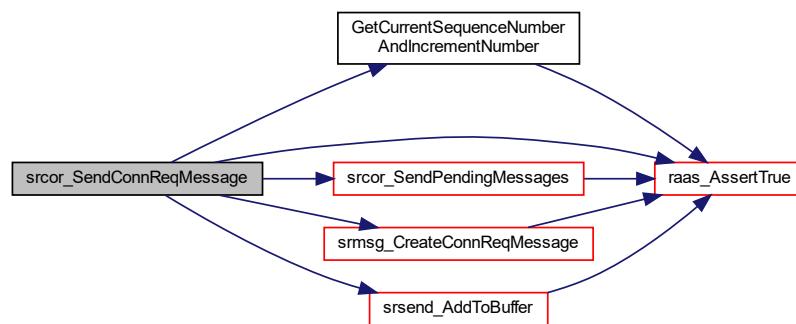
The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-582](#) Send ConnReq Message Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.33 srcor_SendConnRespMessage() `void srcor_SendConnRespMessage (const uint32_t connection_id)`

Create and send a SafRetL connection response message.

This function prepares a new message header that is used to create a new connection response message. This message is then added to the send buffer and send with [srcor_SendPendingMessages](#).

Precondition

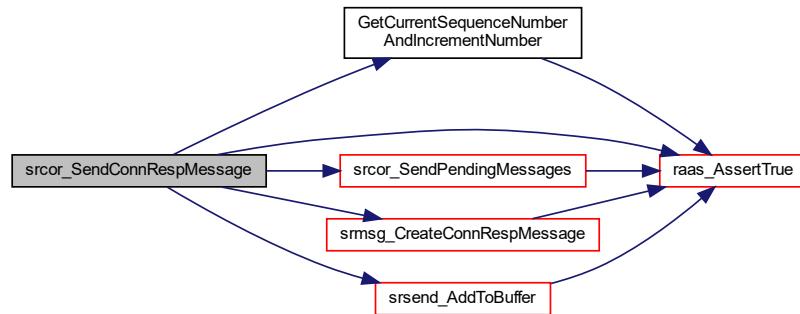
The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-583](#) Send ConnResp Message Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.34 srcor_SendDataMessage() `void srcor_SendDataMessage (const uint32_t connection_id)`

Create and send a SafRetL data message from the temporary buffer.

This function prepares a new message header that is used to create a new data message with the message saved in the temporary buffer. This message is then added to the send buffer and send with [srcor_SendPendingMessages](#). Finally, the flag for a message in the temporary buffer is reset.

Precondition

The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

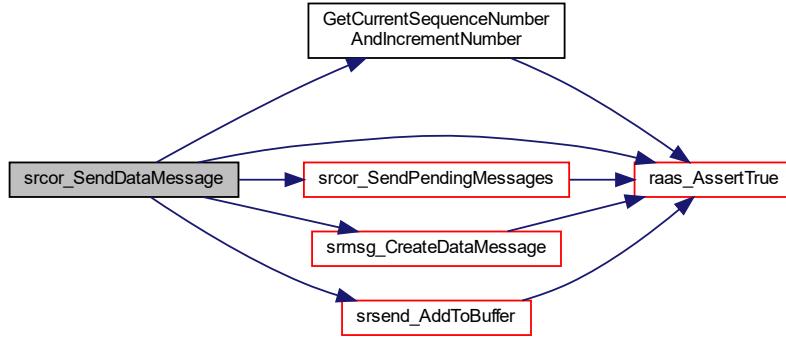
A message must be in the temporary send buffer, otherwise a [radef_kNoMessageToSend](#) fatal error is thrown.

Implements Requirements [RASW-584](#) Send Data Message Function

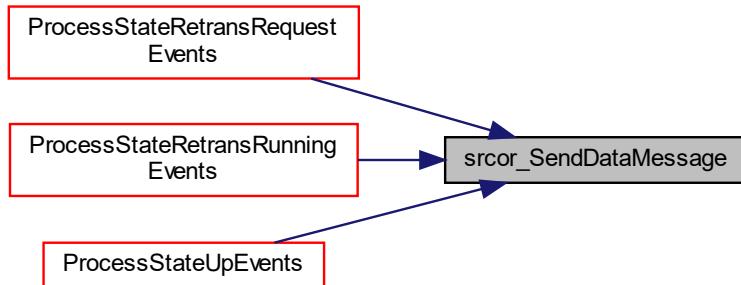
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.35 `srcor_SendDiscReqMessage()` `void srcor_SendDiscReqMessage (`
`const uint32_t connection_id,`
`const sraty_Discreason disconnect_reason)`

Create and send a SafRetL disconnection request message.

This function prepares a new message header that is used to create a disconnection request message. The message is updated with the current timestamp and confirmed sequence number, so it can be send directly ([sradin_SendMessage](#)) without passing through the send buffer. The send buffer of this connection is then initialized ([srsend_InitBuffer](#)) to properly clean all not send messages, since the connection was closed. Finally, the redundancy channel of this connection is closed and a diagnostic notification is send to the application layer (by calling [srcor_CloseRedundancyChannel](#)).

Precondition

The core module must be initialized, otherwise a `raef_kNotInitialized` fatal error is thrown.

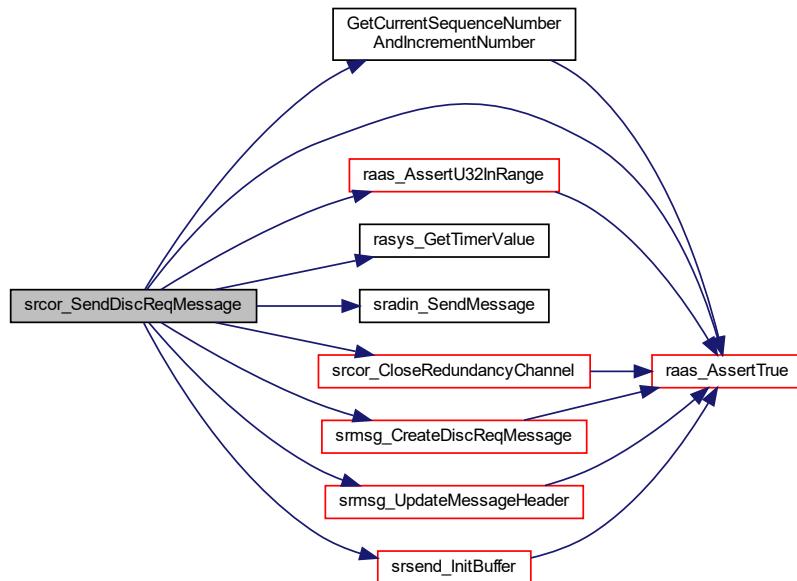
Implements Requirements `RASW-585` Send DiscReq Message Function

`RASW-767` Send Disconnection Request Message Sequence

Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<code>disconnect_reason</code>	Reason for disconnection. Valid range: <code>srtiy_kDiscReasonMin</code> $\leq \text{value} < \text{srtiy_kDiscReasonMax}$.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.36 `srcor_SendHbMessage()` `void srcor_SendHbMessage (`
`const uint32_t connection_id)`

Create and send a SafRetL heartbeat message.

This function creates a new HB message, adds it to the send buffer and send the pending messages with [srcor_SendPendingMessages](#).

Precondition

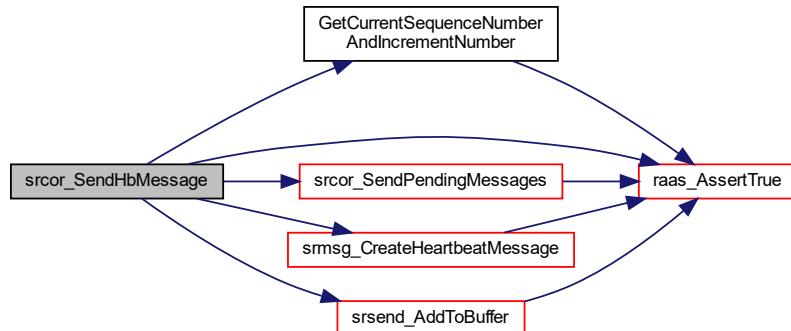
The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-586](#) Send Heartbeat Message Function

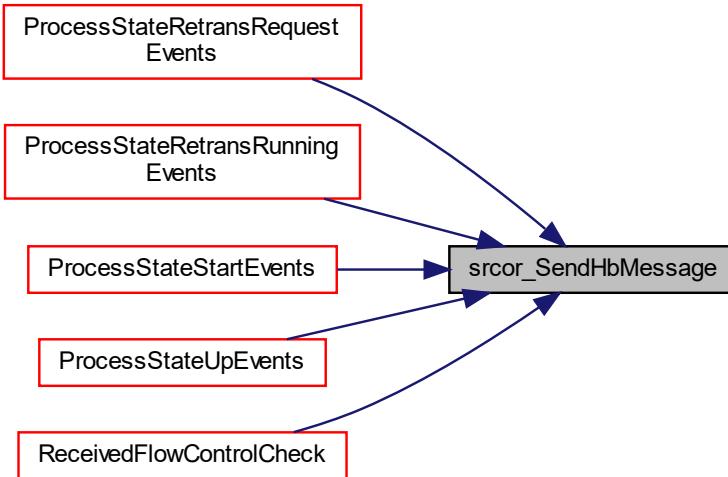
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.37 `srcor_SendPendingMessages()` `void srcor_SendPendingMessages (`
`const uint32_t connection_id)`

Send pending messages from the send buffer, if the flow control allows to send.

This function sends available messages from the send buffer, while there are messages in the send buffer and if the flow control allows to send messages ([SendPendingMessagesWithFlowControlAllowed](#)). While this is true, following steps are performed:

- a message is read from the send buffer
- last send timestamp (`time_stamp_tx`) is updated with the current time
- a [`srtyp_SrMessageHeaderUpdate`](#) structure is prepared
 - set [`srcor_RaStaConnectionData::time_stamp_tx`](#) as timestamp (TS_PDU)
 - in case of a ConnReq message, [`srcor_RaStaConnectionData::confirmed_sequence_number_tx`](#) (CS_T) must be set to 0
 - set [`srcor_RaStaConnectionData::confirmed_sequence_number_tx`](#) (CS_T) as confirmed sequence number (CS_PDU)
 - [`srcor_RaStaConnectionData::last_send_confirmed_sequence_number_tx`](#) is updated with confirmed sequence number (CS_PDU)
- The message is then updated with this [`srtyp_SrMessageHeaderUpdate`](#) structure and finally send with [`sradin_SendMessage`](#).

Precondition

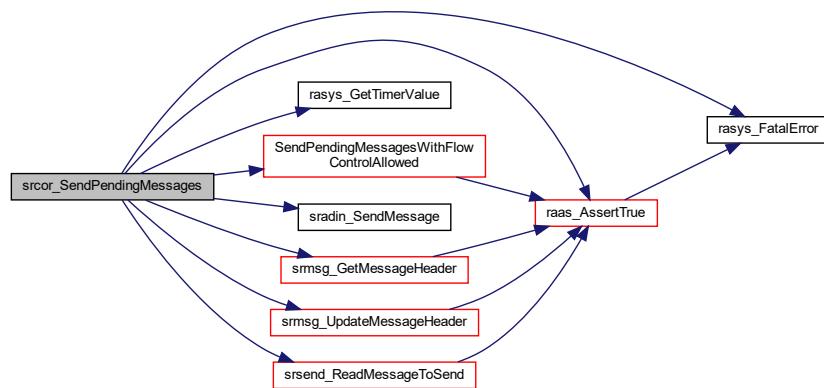
The core module must be initialized, otherwise a [`radef_kNotInitialized`](#) fatal error is thrown.

Implements Requirements [RASW-587](#) Send Pending Messages Function
[RASW-765](#) Send Pending Messages Sequence

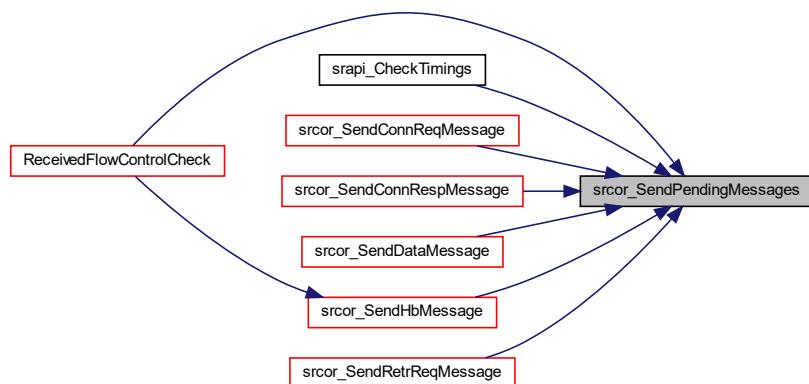
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.38 `srcor_SendRetrReqMessage()` `void srcor_SendRetrReqMessage (const uint32_t connection_id)`

Create and send a SafRetL retransmission request message.

This function creates a new RetrReq message, adds it to the send buffer and send the pending messages with [srcor_SendPendingMessages](#).

Precondition

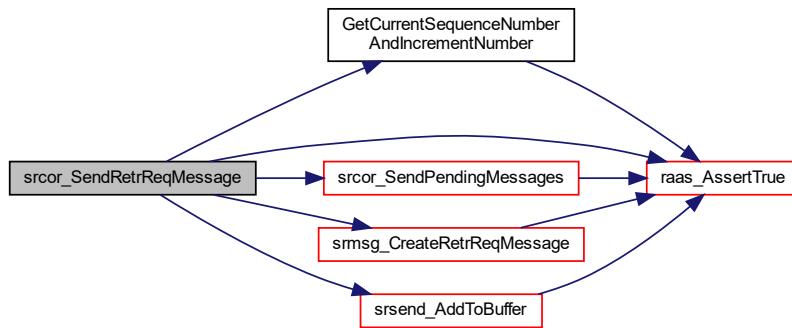
The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-588](#) Send RetrReq Message Function

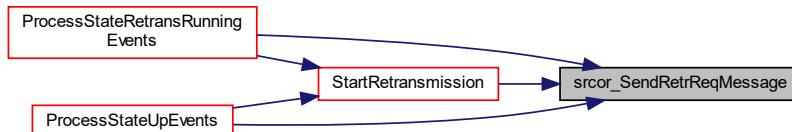
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.39 srcor_SetDiscDetailedReason() `void srcor_SetDiscDetailedReason (const uint32_t connection_id, const uint16_t detailed_disconnect_reason)`

Set detailed reason for disconnection request message.

When a disconnection was requested from the application layer, the passed detailed reason can be stored in the core layer. This is needed to send the disconnection request to the opposite side.

Precondition

The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-589](#) Set Disconnection Detailed Reason Function

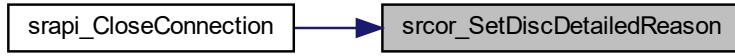
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>detailed_disconnect_reason</i>	Detailed disconnect reason from the application. The full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.40 srcor_SetReceivedMessagePendingFlag() `void srcor_SetReceivedMessagePendingFlag (const uint32_t connection_id)`

Set the message pending flag of a dedicated RaSTA connection.

This function sets the received message pending flag of a dedicated connection. This indicates, that a new message from the RedL is pending to be read.

Precondition

The core module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-590](#) Set Received Message Pending Flag Function

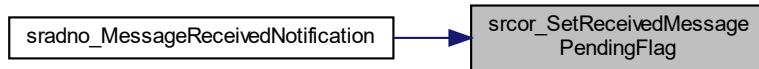
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.41 srcor_UpdateConfirmedRxSequenceNumber() void srcor_UpdateConfirmedRxSequenceNumber()
 (const uint32_t connection_id)

Update confirmed sequence number receive from message in input buffer.

When a message is in the internal input buffer, this function updates the last received confirmed sequence number with the confirmed sequence number received in the last PDU message (CS_R = CS_PDU, is part of action [4] in state event matrix).

Precondition

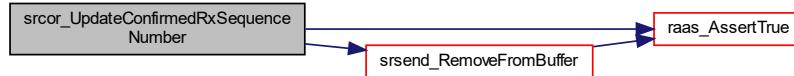
The core module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-591](#) Update Confirmed Rx Sequence Number Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.42 srcor_UpdateConfirmedTxSequenceNumber() void srcor_UpdateConfirmedTxSequenceNumber()
 (const uint32_t connection_id)

Update confirmed sequence number to transmit from the message in the input buffer.

When a message is in the internal input buffer, this function updated the confirmed sequence number to be confirmed with the sequence number of the last received PDU message. (CS_T = SN_PDU, is part of action [1] in state event matrix).

Precondition

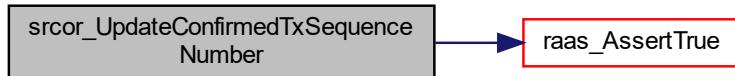
The core module must be initialized, otherwise a [raodef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-592](#) Update Confirmed Tx Sequence Number Function

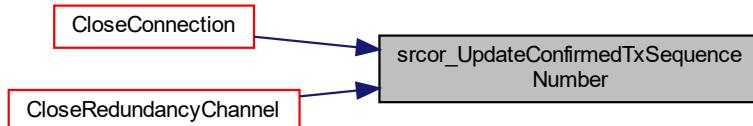
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.43 srcor_WriteMessagePayloadToTemporaryBuffer()

```
void srcor_WriteMessagePayloadToTemporaryBuffer (
    const uint32_t connection_id,
    const uint16_t message_payload_size,
    const uint8_t *const message_payload )
```

Write message payload to temporary buffer for messages to send.

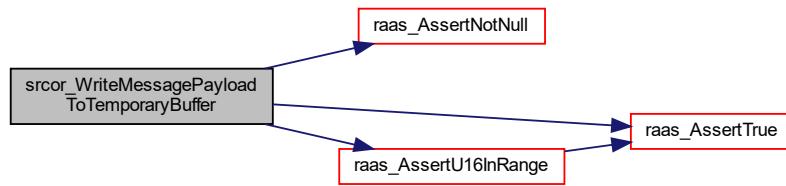
This functions copies the passed message payload structure to the internal temporary buffer and sets the message_in_buffer flag to true. If this function is called and the message_in_buffer flag is still set, a [radef_kInternalError](#) error is thrown.

Implements Requirements [RASW-593](#) Write Message Payload to Temporary Buffer Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>message_payload_size</i>	Size of the message payload [bytes]. Valid range: srcty_kMinSrLayerPayloadDataSize $\leq \text{value} \leq$ RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE .
in	<i>message_payload</i>	Pointer to message payload data. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3 Variable Documentation

4.6.3.1 `srcor_kProtocolVersion` const `srtyp_ProtocolVersion` `srcor_kProtocolVersion` [extern]

Definition of RaSTA protocol version.

Definition of RaSTA protocol version.

4.7 Send Buffer component

Interface of RaSTA SafRetL send buffer module.

Collaboration diagram for Send Buffer component:



Functions

- static void `InitBuffer` (const uint32_t connection_id)

Initialize the send buffer of a dedicated RaSTA connection.
- static void `AddToBuffer` (const uint32_t connection_id, const `srtyp_SrMessage` *const message)

Add a SafRetL message to the send buffer of a dedicated RaSTA connection.
- static void `CopyTempBufferIntoConnectionBuffer` (const uint32_t connection_id)

Copy the temporary buffer into a connection buffer.
- static void `CreateNewDataMsgAndAddToTempBuffer` (const `srtyp_SrMessage` *const sr_message, `srtyp_SrMessageHeaderCreate` *const new_msg_header)

Create a new Data message and add it to the temporary buffer.
- static void `CreateNewRetrDataMsgAndAddToTempBuffer` (const `srtyp_SrMessage` *const sr_message, `srtyp_SrMessageHeaderCreate` *const new_msg_header)

Create a new RetrData message and add it to the temporary buffer.
- static void `CreateNewHbMsgAndAddToTempBuffer` (`srtyp_SrMessageHeaderCreate` *const new_msg_header)

Create a new heartbeat message and add it to the temporary buffer.
- static void `CreateNewRetrReqMsgAndAddToTempBuffer` (`srtyp_SrMessageHeaderCreate` *const new_msg_header)

Create a new RetrReq message and add it to the temporary buffer.
- static void `IncrementSendBufferIndexAndHandleOverflow` (uint16_t *const bufferIndex, const uint16_t increment)

Increment a send buffer index by a increment value and handle overflow.
- void `srsend_Init` (const uint32_t configured_connections)

Initialize all data of the SafRetL send buffer module.
- void `srsend_InitBuffer` (const uint32_t connection_id)

Initialize the send buffer of a dedicated RaSTA connection.
- void `srsend_AddToBuffer` (const uint32_t connection_id, const `srtyp_SrMessage` *const message)

Add a SafRetL message to the send buffer of a dedicated RaSTA connection.
- `radef_RaStaReturnCode srsend_ReadMessageToSend` (const uint32_t connection_id, `srtyp_SrMessage` *const message)

Read a SafRetL message from the send buffer of a dedicated RaSTA connection.
- void `srsend_PreparesBufferForRetr` (const uint32_t connection_id, const uint32_t sequence_number_for_retransmission, const `srtyp_SrMessageHeaderCreate` message_header, uint32_t *const new_current_sequence_number)

Prepare send buffer for retransmission starting with a defined sequence number.
- `radef_RaStaReturnCode srsend_IsSequenceNumberInBuffer` (const uint32_t connection_id, const uint32_t sequence_number)

Checks if a message with a specific sequence number is in the send buffer.
- void `srsend_RemoveFromBuffer` (const uint32_t connection_id, const uint32_t confirmed_sequence_number)

Remove confirmed SafRetL messages from the send buffer from a defined sequence number.
- uint16_t `srsend_GetFreeBufferEntries` (const uint32_t connection_id)

Get the number of free buffer entries.
- uint16_t `srsend_GetUsedBufferEntries` (const uint32_t connection_id)

Get the number of used buffer entries.
- uint16_t `srsend_GetNumberOfMessagesToSend` (const uint32_t connection_id)

Get the number of messages to send from the send buffer.

4.7.1 Detailed Description

Interface of RaSTA SafRetL send buffer module.

Specification of the Send Buffer component of the RaSTA Safety and Retransmission Layer.

This module buffers the messages which are waiting for transmission and sent messages which could be requested for retransmission, since they are not yet confirmed. The send buffer is organized as a FIFO ring buffer. One buffer entry holds a [srtyp_SrMessage](#) struct, which contains a whole SafRetL PDU message.

Implements Requirements	RASW-595 Component sr_send_buffer Overview RASW-518 Safety and Retransmission Layer Safety Integrity Level RASW-520 Error Handling RASW-521 Input Parameter Check
--------------------------------	--

4.7.2 Function Documentation

```
4.7.2.1 AddToBuffer() static void AddToBuffer (
    const uint32_t connection_id,
    const srtyp\_SrMessage *const message ) [static]
```

Add a SafRetL message to the send buffer of a dedicated RaSTA connection.

When there is free space in the buffer, a SafRetL message is added to the buffer. If the buffer is full, a [raodef_kSendBufferFull](#) fatal error message is thrown. After adding the message to the buffer, the position index and used entires are updated. This also lets you add a element to the temporary buffer that sits at the end of the array after the normal connection buffer elements.

Precondition

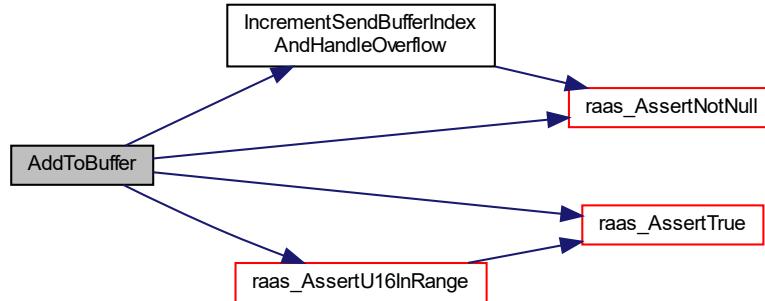
The send buffer module must be initialized, otherwise a [raodef_kNotInitialized](#) fatal error is thrown.

Implements Requirements	RASW-596 Add to Buffer Function
--------------------------------	---

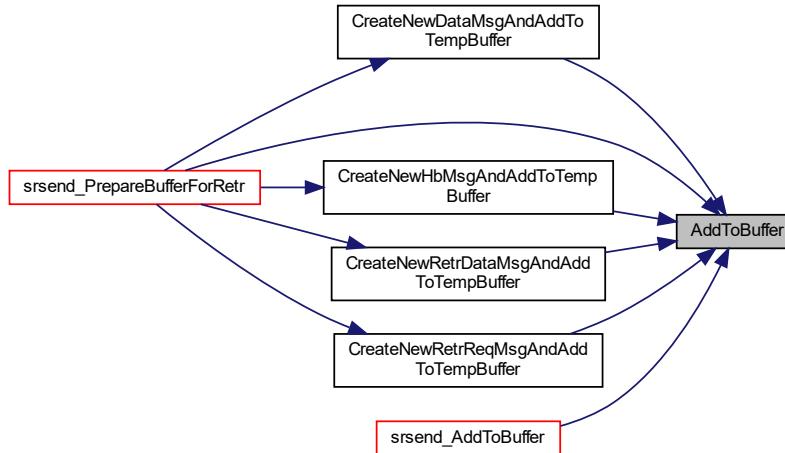
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} \leq \text{configured number of connections}$ (Also includes temporary buffer element).
in	<i>message</i>	Pointer to SafRetL message structure that must be added to the buffer. If the pointer is NULL, a raodef_kInternalError fatal error is thrown. For the message payload the full value range is valid and usable, the message size has a valid range of RADEF_SR_LAYER_MESSAGE_HEADER_SIZE $\leq \text{value} \leq$ RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE .

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.2 CopyTempBufferIntoConnectionBuffer() static void CopyTempBufferIntoConnectionBuffer (const uint32_t connection_id) [static]

Copy the temporary buffer into a connection buffer.

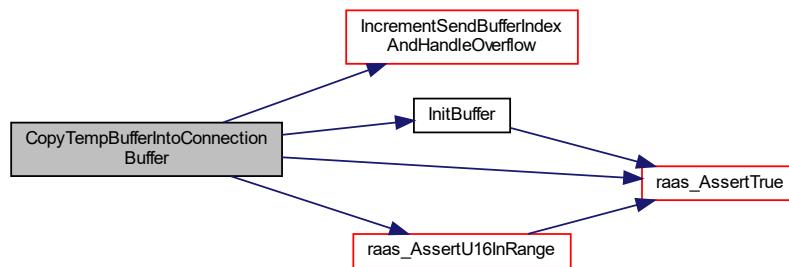
This internal function initializes the buffer of a connection and then copies the temporary buffer, which was prepared in advance for a retransmission, into this empty connection buffer.

Implements Requirements RASW-603 Prepare Buffer for Retransmission Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.3 CreateNewDataMsgAndAddToTempBuffer() static void CreateNewDataMsgAndAddToTempBuffer()
 (const `srtyp_SrMessage` *const `sr_message`,
`srtyp_SrMessageHeaderCreate` *const `new_msg_header`) [static]

Create a new Data message and add it to the temporary buffer.

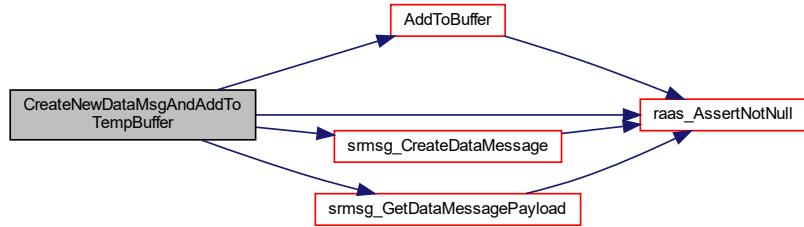
This internal function creates a new Data message based on the passed Data message. It is then added to the temporary buffer and the sequence number in the header is incremented.

Implements Requirements RASW-603 Prepare Buffer for Retransmission Function

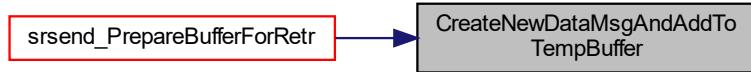
Parameters

in	<i>sr_message</i>	Pointer to the input data or retrData message. If the pointer is NULL, a <code>radef_kInternalError</code> fatal error is thrown.
in, out	<i>new_msg_header</i>	Pointer to the header data to create a new message. The header contains also the current sequence number from which the new message must continue and which at the end of the function will be incremented by one. If the pointer is NULL, a <code>radef_kInternalError</code> fatal error is thrown.
Generated by Doxygen		

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.4 CreateNewHbMsgAndAddToTempBuffer() `static void CreateNewHbMsgAndAddToTempBuffer (srtyp_SrMessageHeaderCreate *const new_msg_header) [static]`

Create a new heartbeat message and add it to the temporary buffer.

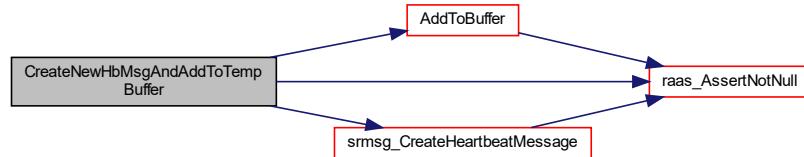
This internal function creates a new heartbeat message. It is then added to the temporary buffer and the sequence number in the header is incremented.

Implements Requirements [RASW-603](#) Prepare Buffer for Retransmission Function

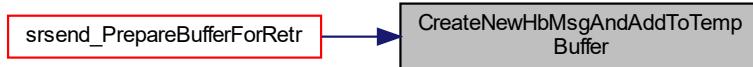
Parameters

<code>in, out</code>	<code>new_msg_header</code>	Pointer to the header data to create a new message. The header contains also the current sequence number from which the new message must continue and which at the end of the function will be incremented by one. If the pointer is NULL, a raef_kInternalError fatal error is thrown.
----------------------	-----------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.5 CreateNewRetrDataMsgAndAddToTempBuffer() static void CreateNewRetrDataMsgAndAddToTempBuffer (

```

        const srtyp_SrMessage *const sr_message,
        srtyp_SrMessageHeaderCreate *const new_msg_header ) [static]

```

Create a new RetrData message and add it to the temporary buffer.

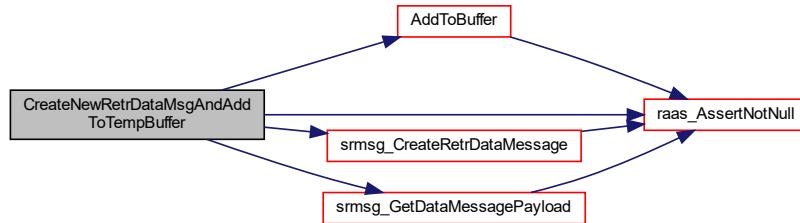
This internal function creates a new RetrData message based on the passed Data or RetrData message. It is then added to the temporary buffer and the sequence number in the header is incremented.

Implements Requirements RASW-603 Prepare Buffer for Retransmission Function

Parameters

in	<code>sr_message</code>	Pointer to the input data or retrData message. If the pointer is NULL, a <code>radef_kInternalError</code> fatal error is thrown.
in, out	<code>new_msg_header</code>	Pointer to the header data to create a new message. The header contains also the current sequence number from which the new message must continue and which at the end of the function will be incremented by one. If the pointer is NULL, a <code>radef_kInternalError</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.6 CreateNewRetrReqMsgAndAddToTempBuffer() static void CreateNewRetrReqMsgAndAddToTempBuffer (
 srtyp_SrMessageHeaderCreate *const new_msg_header) [static]

Create a new RetrReq message and add it to the temporary buffer.

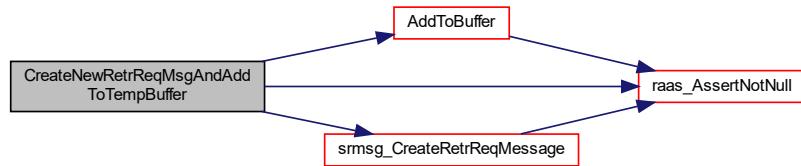
This internal function creates a new RetrReq message. It is then added to the temporary buffer and the sequence number in the header is incremented.

Implements Requirements RASW-603 Prepare Buffer for Retransmission Function

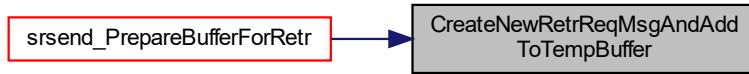
Parameters

in, out	<code>new_msg_header</code>	Pointer to the header data to create a new message. The header contains also the current sequence number from which the new message must continue and which at the end of the function will be incremented by one. If the pointer is NULL, a <code>radef_KInternalError</code> fatal error is thrown.
---------	-----------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.7 IncrementSendBufferIndexAndHandleOverflow() static void IncrementSendBufferIndexAndHandleOverflow (uint16_t *const bufferIndex, const uint16_t increment) [static]

Increment a send buffer index by a increment value and handle overflow.

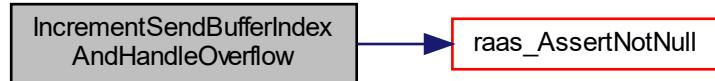
This internal function increments a passed buffer index by a specified increment and handles a possible overflow by exceeding the maximum buffer size [RADEF_SEND_BUFFER_SIZE](#). In case of an overflow, the maximum value is subtracted from the index.

Implements Requirements [RASW-604](#) Read Message to Send Function
[RASW-603](#) Prepare Buffer for Retransmission Function
[RASW-605](#) Remove from Buffer Function

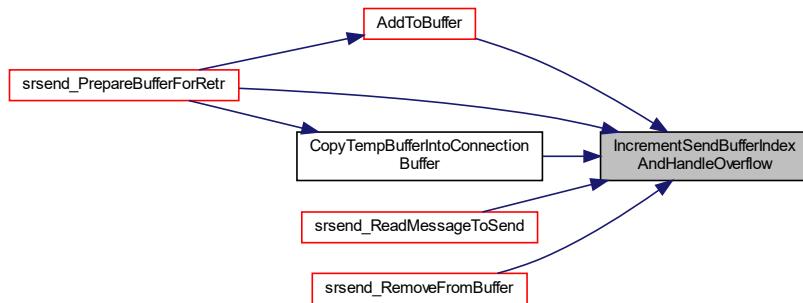
Parameters

in	<i>bufferIndex</i>	Pointer to the buffer index. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	<i>increment</i>	Index increment value. The full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.8 InitBuffer() `static void InitBuffer (const uint32_t connection_id) [static]`

Initialize the send buffer of a dedicated RaSTA connection.

This function initializes the buffer of a given RaSTA connection. It resets all properties of the buffer (read, write, remove index and used entries) and also sets the message length of all elements in the buffer to 0 and clears the already sent flag. This also lets you initialize the temporary buffer that sits at the end of the array after the normal connection buffer elements.

Precondition

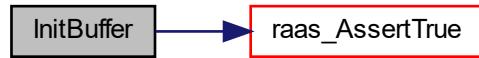
The send buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-601](#) Init Buffer Function

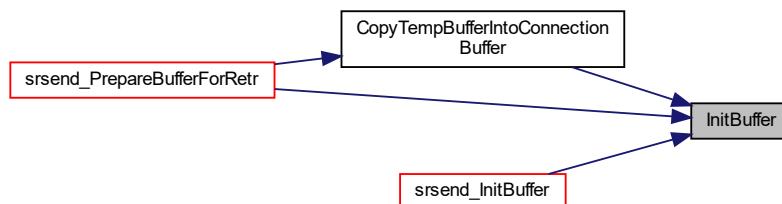
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} \leq \text{configured number of connections}$. (Also includes temporary buffer element).
----	----------------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.9 srsend_AddToBuffer() void srsend_AddToBuffer (const uint32_t connection_id, const srtyp_SrMessage *const message)

Add a SafRetL message to the send buffer of a dedicated RaSTA connection.

When there is free space in the buffer, a SafRetL message is added to the buffer. If the buffer is full, a [radef_kSendBufferFull](#) fatal error message is thrown. After adding the message to the buffer, the position index and used entires are updated.

Precondition

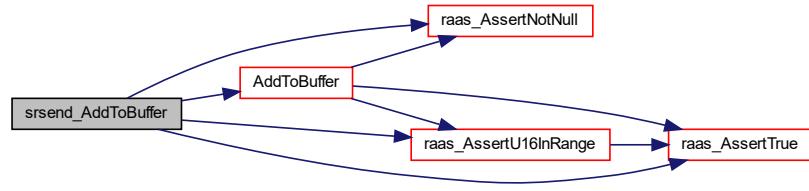
The send buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-596](#) Add to Buffer Function

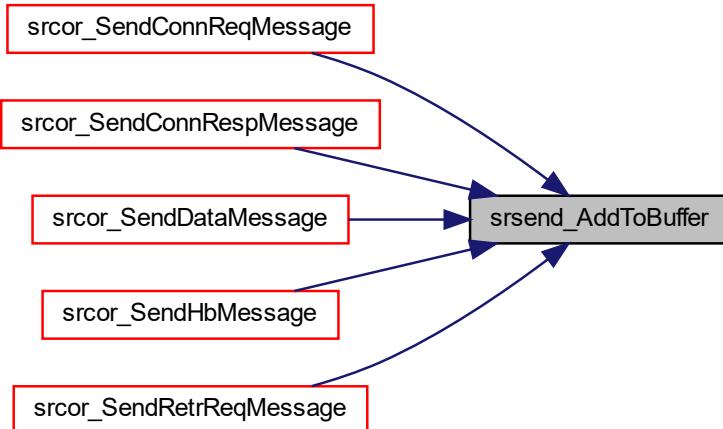
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>message</i>	Pointer to SafRetL message structure that must be added to the buffer. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown. For the message payload the full value range is valid and usable, the message size has a valid range of RADEF_SR_LAYER_MESSAGE_HEADER_SIZE $\leq \text{value} \leq$ RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE .

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.10 srsend_GetFreeBufferEntries() `uint16_t srsend_GetFreeBufferEntries (const uint32_t connection_id)`

Get the number of free buffer entries.

This function returns the amount of free entries in the send buffer of a given connection.

Precondition

The send buffer module must be initialized, otherwise a `raodef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-598](#) Get Free Buffer Entries Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

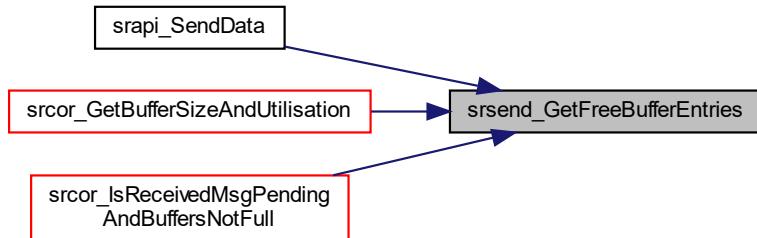
Returns

number of free entries in the send buffer [messages]

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.11 srsend_GetNumberOfMessagesToSend() `uint16_t srsend_GetNumberOfMessagesToSend (const uint32_t connection_id)`

Get the number of messages to send from the send buffer.

This function returns the number of not yet send messages in the send buffer of a given connection.

Precondition

The send buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-599](#) Get Number of Messages to Send Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

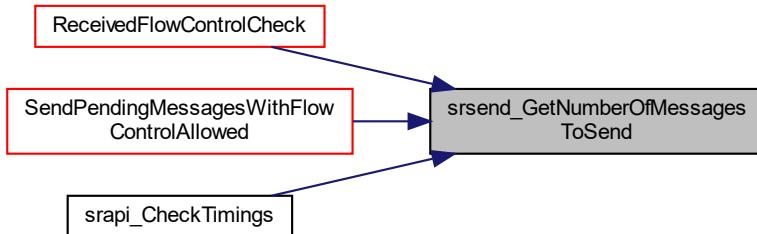
Returns

number of messages to send in the send buffer [messages]

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.12 srsend_GetUsedBufferEntries() `uint16_t srsend_GetUsedBufferEntries (`
`const uint32_t connection_id)`

Get the number of used buffer entries.

This function returns the amount of used entries in the send buffer of a given connection.

Precondition

The send buffer module must be initialized, otherwise a [raderf_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-597](#) Get Used Buffer Entries Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

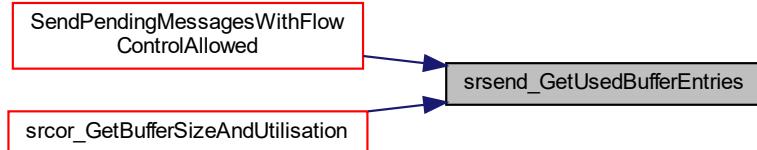
Returns

number of used entries in the send buffer [messages]

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.13 srsend_Init() `void srsend_Init (const uint32_t configured_connections)`

Initialize all data of the SafRetL send buffer module.

This function is used to initialize the send buffer module. It saves the passed number of connections. For all configured connections, the `srsend_InitBuffer` function is called to properly initialize the buffer for all configured connections. A fatal error is raised, if this function is called multiple times.

Precondition

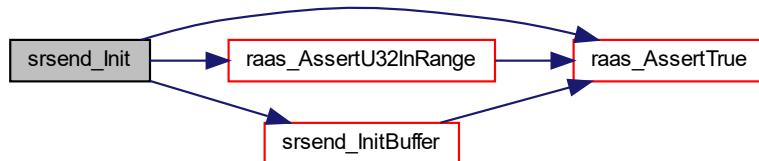
The send buffer module must not be initialized, otherwise a `radef_kAlreadyInitialized` fatal error is thrown.

Implements Requirements [RASW-600](#) Init sr_send_buffer Function

Parameters

in	<i>configured_connections</i>	Number of configured RaSTA connections. Valid range: $1 \leq \text{value} \leq \text{RADEF_MAX_NUMBER_OF_RSTA_CONNECTIONS}$.
----	-------------------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.14 srsend_InitBuffer() `void srsend_InitBuffer (const uint32_t connection_id)`

Initialize the send buffer of a dedicated RaSTA connection.

This function initializes the buffer of a given RaSTA connection. It resets all properties of the buffer (read, write, remove index and used entries) and also sets the message length of all elements in the buffer to 0 and clears the already sent flag.

Precondition

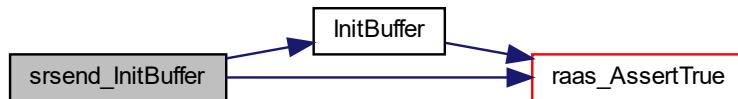
The send buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-601](#) Init Buffer Function

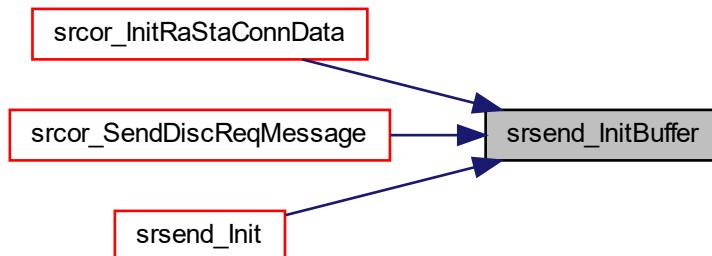
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.15 srsend_IsSequenceNumberInBuffer() *radef_RaStaReturnCode* srsend_IsSequenceNumberInBuffer (

```

        const uint32_t connection_id,
        const uint32_t sequence_number )

```

Checks if a message with a specific sequence number is in the send buffer.

This function checks if a specific sequence number is present in the send buffer of a given RaSTA connection and returns *radef_kNoError* if the number is found or *radef_kInvalidSequenceNumber* if its not found.

Precondition

The send buffer module must be initialized, otherwise a *radef_kNotInitialized* fatal error is thrown.

Implements Requirements *RASW-602* Is Sequence Number in Buffer Function

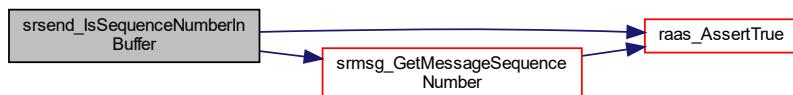
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
in	<i>sequence_number</i>	Sequence number to check if present in send buffer. The full value range is valid and usable.

Returns

radef_kNoError -> sequence number found in the send buffer
 radef_kInvalidSequenceNumber -> sequence number not found in the send buffer

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.16 srsend_PrepBufferForRetr() void srsend_PrepBufferForRetr (

```

const uint32_t connection_id,
const uint32_t sequence_number_for_retransmission,
const srtyp_SrMessageHeaderCreate message_header,
uint32_t *const new_current_sequence_number )
  
```

Prepare send buffer for retransmission starting with a defined sequence number.

This function prepares the send buffer for a retransmission of all messages after a given sequence number. For this, the following steps are done:

- remove messages which must not be retransmitted (all other than Data and RetrData)
- add RetrResp message to the send buffer
- transform requested already sent Data & RetrData messages

- while still not yet send RetrData is available, add it for retransmission. As soon as an other message type is added, RetrData is then forbidden
 - add a heartbeat message after the retransmission, if no data message is pending in the buffer to indicate the end of the retransmission
 - the remaining not yet send messages can be added to the buffer. (HB messages must be filtered from beeing send. Only RetrReq and Data are valid, other message types should not be in the buffer anymore)

Precondition

The send buffer module must be initialized, otherwise a `raef_kNotInitialized` fatal error is thrown.

All messages before and the requested sequence number must be confirmed before calling this method.

Sequence number for retransmission must be in the buffer, otherwise a `raodef_klInvalidSequenceNumber` fatal error is thrown.

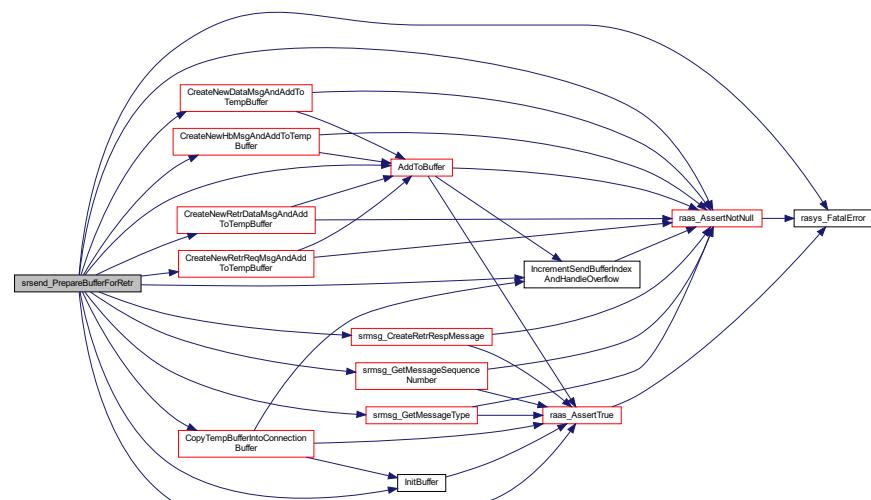
It is assumed, that the buffer elements are in proper order with ascending sequence number and without gaps in the sequence numbers.

Implements Requirements RASW-603 Prepare Buffer for Retransmission Function

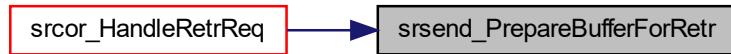
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>sequence_number_for_retransmission</i>	Last good sequence number after which the retransmission must be started. The full value range is valid and usable.
in	<i>message_header</i>	Message header data to create a new message. The message header contains also the current sequence number from which the new message must continue.
out	<i>new_current_sequence_number</i>	New sequence number for new messages after reworking the send buffer for the retransmission. If the pointer is NULL, a raodef_klInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.17 srsend_ReadMessageToSend() [radef_RaStaReturnCode](#) srsend_ReadMessageToSend (const uint32_t connection_id, [srtyp_SrMessage](#) *const message)

Read a SafRetL message from the send buffer of a dedicated RaSTA connection.

When there are messages in the buffer, the oldest SafRetL message is read from the buffer, saved into the passed structure, the read position pointer is updated and a [radef_kNoError](#) returned. If the buffer is empty, a [radef_kNoMessageToSend](#) is returned.

Precondition

The send buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-604](#) Read Message to Send Function

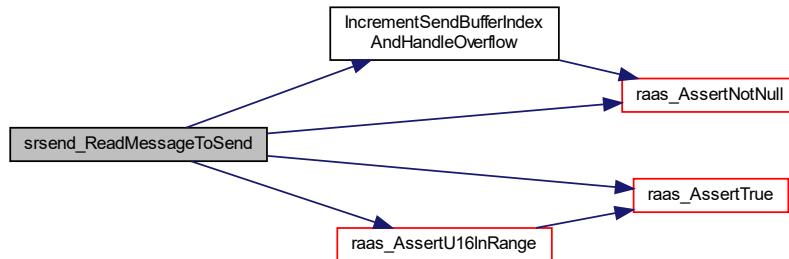
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
out	<i>message</i>	Pointer to SafRetL message structure, where the message must be saved. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Returns

`radef_kNoError` -> message successfully read from the buffer
`radef_kNoMessageToSend` -> no message in the buffer

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.18 srsend_RemoveFromBuffer() `void srsend_RemoveFromBuffer (`
`const uint32_t connection_id,`
`const uint32_t confirmed_sequence_number)`

Remove confirmed SafRetL messages from the send buffer from a defined sequence number.

This function removes the message with a specific sequence number and all previous messages from the send buffer of a dedicated RaSTA connection. If the passed sequence number or its predecessors are not found, nothing is done.

Precondition

The send buffer module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

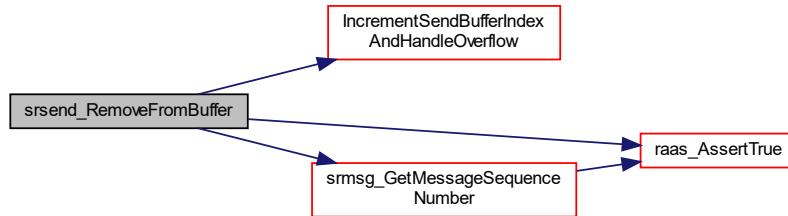
Ascending sequence number order of buffer elements is guaranteed by `srsend_AddToBuffer` and `srsend_PreparesBufferForRetr`.

Implements Requirements [RASW-605 Remove from Buffer Function](#)

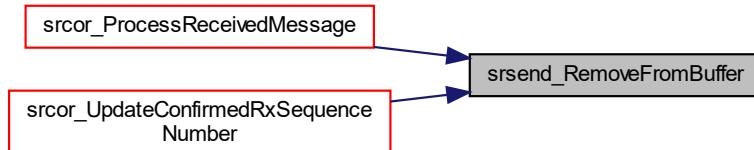
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<i>confirmed_sequence_number</i>	Confirmed sequence number to be removed from send buffer. The full value range is valid and usable.

Here is the call graph for this function:



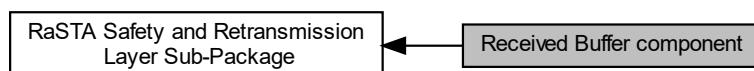
Here is the caller graph for this function:



4.8 Received Buffer component

Interface of RaSTA SafRetL received buffer module.

Collaboration diagram for Received Buffer component:



Functions

- static void [IncrementReceivedBufferIndexAndHandleOverflow](#) (uint16_t *const bufferIndex, const uint16_t increment)

Increment a received buffer index by a increment value and handle overflow.
- void [srrece_Init](#) (const uint32_t configured_connections, const uint16_t configured_n_send_max)

Initialize the SafRetL received buffer module.
- void [srrece_InitBuffer](#) (const uint32_t connection_id)

Initialize the received buffer of a dedicated RaSTA connection.
- void [srrece_AddToBuffer](#) (const uint32_t connection_id, const [srtyp_SrMessagePayload](#) *const message_payload)

Add a SafRetL message to the received buffer of a dedicated RaSTA connection. A fatal error is raised, if the buffer is full.
- [radef_RaStaReturnCode srrece_ReadFromBuffer](#) (const uint32_t connection_id, [srtyp_SrMessagePayload](#) *const message_payload)

Read and remove a SafRetL message payload from the received buffer of a dedicated RaSTA connection.
- uint32_t [srrece_GetPayloadSizeOfNextMessageToRead](#) (const uint32_t connection_id)

Get the payload size of the next message that is read from a dedicated RaSTA connection.
- uint16_t [srrece_GetFreeBufferEntries](#) (const uint32_t connection_id)

Get the number of free buffer entries.
- uint16_t [srrece_GetUsedBufferEntries](#) (const uint32_t connection_id)

Get the number of used buffer entries.

4.8.1 Detailed Description

Interface of RaSTA SafRetL received buffer module.

Specification of the Received Buffer component of the RaSTA Safety and Retransmission Layer.

This module buffers the payload of successfully received SafRetL messages, for the read from the application layer. The received buffer is organized as a FIFO ring buffer. One buffer entry holds a [srtyp_SrMessagePayload](#) struct, which contains the payload of SafRetL PDU message.

Implements Requirements [RASW-607](#) Component sr_received_buffer Overview
[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level
[RASW-520](#) Error Handling
[RASW-521](#) Input Parameter Check

4.8.2 Function Documentation

4.8.2.1 IncrementReceivedBufferIndexAndHandleOverflow() static void IncrementReceivedBufferIndexAndHandleOverflow (
 uint16_t *const bufferIndex,
 const uint16_t increment) [static]

Increment a received buffer index by a increment value and handle overflow.

This internal function increments a passed buffer index by a specified increment and handles a possible overflow by exceeding the maximum buffer size [srrece_n_send_max](#). In case of an overflow, the index is set to 0.

Implements Requirements [RASW-608](#) Add to Buffer Function
[RASW-613](#) Read from Buffer Function

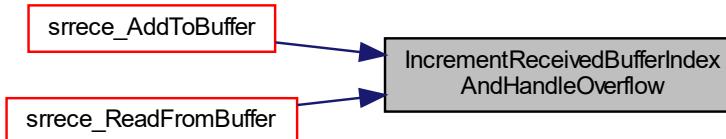
Parameters

in	<i>bufferIndex</i>	Pointer to the buffer index. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	<i>increment</i>	Index increment value. The full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.8.2.2 srrece_AddToBuffer() void srrece_AddToBuffer (
    const uint32_t connection_id,
    const srtyp\_SrMessagePayload *const message_payload )
  
```

Add a SafRetL message to the received buffer of a dedicated RaSTA connection. A fatal error is raised, if the buffer is full.

When there is free space in the buffer, a SafRetL message is added to the buffer. If the buffer is full, a [radef_kReceiveBufferFull](#) fatal error message is thrown. After adding the message to the buffer, the position pointer and used entries are updated.

Precondition

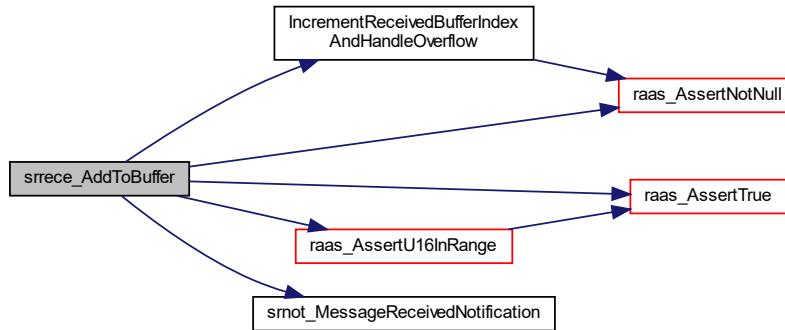
The received buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-608](#) Add to Buffer Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
in	<i>message_payload</i>	Pointer to message payload structure that must be added to the buffer. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown. For the message payload the full value range is valid and usable, the message payload size has a valid range of srcty_kMinSrLayerPayloadDataSize $\leq \text{value} \leq \text{RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE}$.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.3 `srrece_GetFreeBufferEntries()` `uint16_t srrece_GetFreeBufferEntries (const uint32_t connection_id)`

Get the number of free buffer entries.

This function returns the amount of free entries in the received buffer of a given connection.

Precondition

The received buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-610](#) Get Free Buffer Entries Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

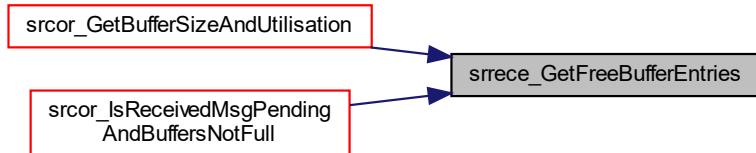
Returns

Number of free entries in the received buffer [messages]

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.4 srrece_GetPayloadSizeOfNextMessageToRead() `uint32_t srrece_GetPayloadSizeOfNextMessageToRead (const uint32_t connection_id)`

Get the payload size of the next message that is read from a dedicated RaSTA connection.

This function returns the payload size of the next message that can be read from a specific connection. If there is no message to be read, 0 will be returned.

Precondition

The received buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-823](#) Get Payload Size of Next Message To Read Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

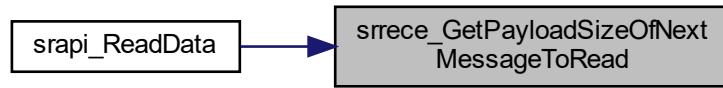
Returns

uint32_t -> Size of the payload of the next message to read [bytes]

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.5 srrece_GetUsedBufferEntries() uint16_t srrece_GetUsedBufferEntries (const uint32_t connection_id)

Get the number of used buffer entries.

This function returns the amount of used entries in the received buffer of a given connection.

Precondition

The received buffer module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-609](#) Get Used Buffer Entries Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

Returns

Number of used entries in the received buffer [messages]

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.6 srrece_Init() void srrece_Init (
 const uint32_t configured_connections,
 const uint16_t configured_n_send_max)

Initialize the SafRetL received buffer module.

This function is used to initialize the received buffer module. It saves the passed number of connections and the configured size of the buffer (NsendMax). For all configured connections, the [srrece_InitBuffer](#) function is called to properly initialize the buffer for all configured connections. A fatal error is raised, if this function is called multiple times.

Precondition

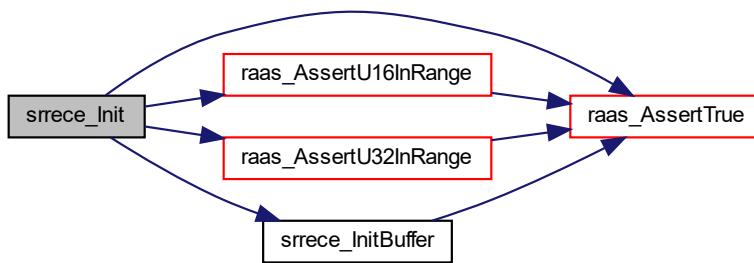
The received buffer module must not be initialized, otherwise a [radef_kAlreadyInitialized](#) fatal error is thrown.

Implements Requirements [RASW-611](#) Init sr_received_buffer Function

Parameters

in	<i>configured_connections</i>	Number of configured RaSTA connections. Valid range: $1 \leq \text{value} \leq \text{RADEF_MAX_NUMBER_OF_RSTA_CONNECTIONS}$.
in	<i>configured_n_send_max</i>	Configured receive buffer size [messages]. Valid range: $\text{srcty_kMinNSendMax} \leq \text{value} \leq \text{RADEF_MAX_N_SEND_MAX}$.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.7 `srrece_InitBuffer()` `void srrece_InitBuffer (`
`const uint32_t connection_id)`

Initialize the received buffer of a dedicated RaSTA connection.

This function initializes the buffer of a given RaSTA connection. It resets all properties of the buffer (read, write index and used entries) and also sets the message length of all elements in the buffer to 0.

Precondition

The received buffer module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-612](#) Init Buffer Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	----------------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.8 srrece_ReadFromBuffer() `radef_RaStaReturnCode srrece_ReadFromBuffer (`
`const uint32_t connection_id,`
`srtyp_SrMessagePayload *const message_payload)`

Read and remove a SafRetL message payload from the received buffer of a dedicated RaSTA connection.

When there are messages in the buffer, the oldest SafRetL message is read from the buffer, saved into the passed structure, the position pointers & used entries are updated and a `radef_kNoError` returned. If the buffer is empty, a `radef_kNoMessageReceived` is returned.

Precondition

The received buffer module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements RASW-613 Read from Buffer Function

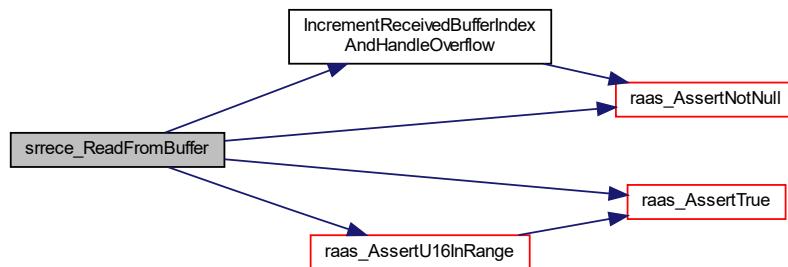
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
out	<i>message_payload</i>	Pointer to SafRetL message payload structure, where the message must be saved. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

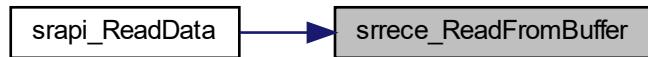
Returns

`radef_kNoError` -> Message successfully read from the buffer.
`radef_kNoMessageReceived` -> No message payload in the buffer.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9 Messages component

Interface of RaSTA SafRetL messages module.

Collaboration diagram for Messages component:



Functions

- static bool `IsMessageTypeValid (srtyp_SrMessageType message_type)`
Checks if a message type is valid or not.
- static void `SetUint16InMessage (const uint16_t position, const uint16_t data, srtyp_SrMessage *const sr_message)`
Set a Uint16 at a specific position in a message.
- static void `SetUint32InMessage (const uint16_t position, const uint32_t data, srtyp_SrMessage *const sr_message)`
Set a Uint32 at a specific position in a message.
- static void `SetUint64InMessage (const uint16_t position, const uint64_t data, srtyp_SrMessage *const sr_message)`
Set a Uint64 at a specific position in a message.
- static void `SetMessageHeaderInMessage (const uint16_t message_length, const uint16_t message_type, const srtyp_SrMessageHeaderCreate message_header, srtyp_SrMessage *const sr_message)`
Set the message header data in a message.
- static void `SetProtocolVersionInMessage (const srtyp_ProtocolVersion protocol_version, srtyp_SrMessage *const sr_message)`
Set the protocol version in a message.
- static void `SetPayloadDataInMessage (const uint16_t position, const srtyp_SrMessagePayload *const message_payload, srtyp_SrMessage *const sr_message)`
Set the payload data in a message.
- static uint16_t `GetUint16FromMessage (const srtyp_SrMessage *const sr_message, const uint16_t position)`
Get a Uint16 from a specific position in a message.
- static uint32_t `GetUint32FromMessage (const srtyp_SrMessage *const sr_message, const uint16_t position)`
Get a Uint32 from a specific position in a message.
- static uint16_t `GetSafetyCodeLength (void)`
Get the length of the configured safety code.
- void `srmsg_Init (const srcty_SafetyCodeType configured_safety_code_type, const srcty_Md4InitValue configured_md4_initial_value)`
Initialize SafRetL messages module.
- void `srmsg_CreateConnReqMessage (const srtyp_SrMessageHeaderCreate message_header, const srtyp_ProtocolVersion protocol_version, const uint16_t n_send_max, srtyp_SrMessage *const sr_message)`
Create a new SafRetL connection request message.
- void `srmsg_CreateConnRespMessage (const srtyp_SrMessageHeaderCreate message_header, const srtyp_ProtocolVersion protocol_version, const uint16_t n_send_max, srtyp_SrMessage *const sr_message)`
Create a new SafRetL connection response message.
- void `srmsg_CreateDataMessage (const srtyp_SrMessageHeaderCreate message_header, const srtyp_SrMessagePayload *const message_payload, srtyp_SrMessage *const sr_message)`
Create a new SafRetL data message.
- void `srmsg_CreateRetrDataMessage (const srtyp_SrMessageHeaderCreate message_header, const srtyp_SrMessagePayload *const message_payload, srtyp_SrMessage *const sr_message)`
Create a new SafRetL retransmitted data message.
- void `srmsg_CreateRetrReqMessage (const srtyp_SrMessageHeaderCreate message_header, srtyp_SrMessage *const sr_message)`
Create a new SafRetL retransmission request message.
- void `srmsg_CreateRetrRespMessage (const srtyp_SrMessageHeaderCreate message_header, srtyp_SrMessage *const sr_message)`
Create a new SafRetL retransmission response message.
- void `srmsg_CreateHeartbeatMessage (const srtyp_SrMessageHeaderCreate message_header, srtyp_SrMessage *const sr_message)`
Create a new SafRetL heartbeat message.

- void `srmsg_CreateDiscReqMessage` (const `srtyp_SrMessageHeaderCreate` message_header, const `uint16_t` detailed_reason, const `srtyp_DiscReason` reason, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL disconnection request message.
- void `srmsg_UpdateMessageHeader` (const `srtyp_SrMessageHeaderUpdate` message_header_update, `srtyp_SrMessage` *const sr_message)

Update a SafRetL message header and calculate the safety code to prepare the message for sending.
- `radef_RaStaReturnCode srmsg_CheckMessage` (const `srtyp_SrMessage` *const sr_message)

Check MD4, message type and message size of a SafRetL PDU message.
- void `srmsg_GetMessageHeader` (const `srtyp_SrMessage` *const sr_message, `srtyp_SrMessageHeader` *const message_header)

Get the header of a SafRetL PDU message.
- `srtyp_SrMessageType srmsg_GetMessageType` (const `srtyp_SrMessage` *const sr_message)

Get the message type of a SafRetL PDU message.
- `uint32_t srmsg_GetMessageSequenceNumber` (const `srtyp_SrMessage` *const sr_message)

Get the sequence number of a SafRetL PDU message.
- void `srmsg_GetDataMessagePayload` (const `srtyp_SrMessage` *const sr_message, `srtyp_SrMessagePayload` *const message_payload)

Get the payload of a SafRetL data or retransmitted data message.
- void `srmsg_GetConnMessageData` (const `srtyp_SrMessage` *const sr_message, `srtyp_ProtocolVersion` *const protocol_version, `uint16_t` *const n_send_max)

Get the data of a SafRetL connection request or connection response message.
- void `srmsg_GetDiscMessageData` (const `srtyp_SrMessage` *const sr_message, `uint16_t` *const detailed_reason, `srtyp_DiscReason` *const reason)

Get the data of a SafRetL disconnection request message.

4.9.1 Detailed Description

Interface of RaSTA SafRetL messages module.

Specification of the Messages component of the RaSTA Safety and Retransmission Layer.

This module provides all needed functionality for SafRetL messages. This contains the following:

- validate a message
- create all different types of SafRetL messages
- update header information of a SafRetL message
- extract header or specific message type informations/data

Implements Requirements [RASW-615](#) Component sr_messages Overview

[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

[RASW-520](#) Error Handling

[RASW-521](#) Input Parameter Check

4.9.2 Function Documentation

4.9.2.1 GetSafetyCodeLength() static uint16_t GetSafetyCodeLength (void) [static]

Get the length of the configured safety code.

This internal function returns the byte length of the configured safety code.

Implements Requirements RASW-168 Safety Code

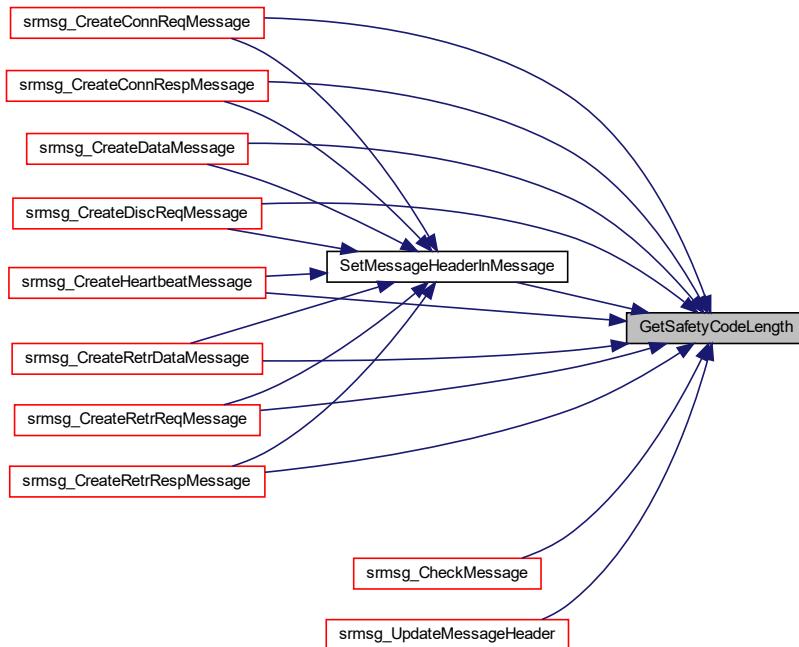
Returns

uint16_t length of the safety code [bytes].

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.9.2.2 GetUint16FromMessage() static uint16_t GetUint16FromMessage (
    const srtyp_SrMessage *const sr_message,
    const uint16_t position ) [static]
```

Get a Uint16 from a specific position in a message.

This internal function extracts a Uint16 byte by byte from a given position in the little endian format message. If the Uint16 extends over the size of the message from the given start position (position + Uint16 byte size > message size), a [radef_kInternalError](#) fatal error message is thrown.

Implements Requirements [RASW-157](#) Endian Definition

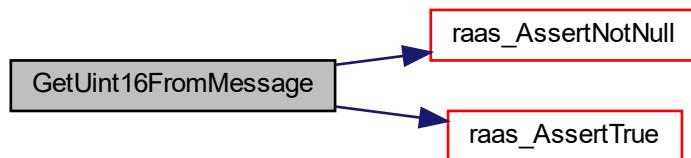
Parameters

in	<i>sr_message</i>	Pointer to a message, from where a variable must be extracted. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	<i>position</i>	Start position where the variable must be read in the message [bytes].

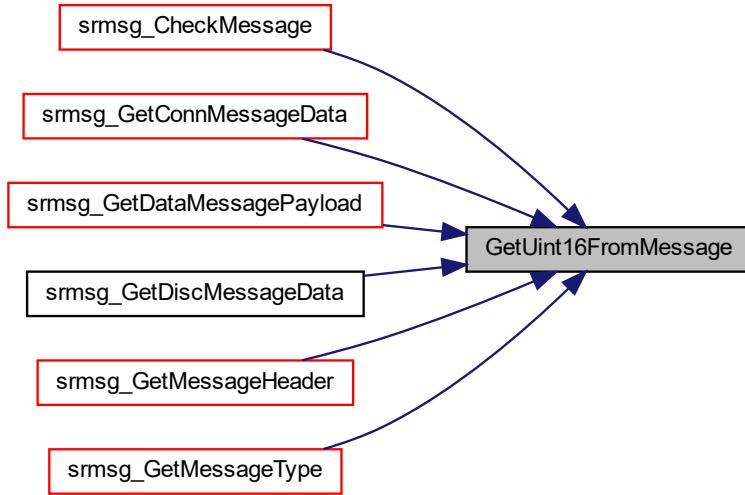
Returns

uint16_t Read variable from the specified start position.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.3 GetUint32FromMessage() static uint32_t GetUint32FromMessage (

```

const srtyp_SrMessage *const sr_message,
const uint16_t position ) [static]
```

Get a Uint32 from a specific position in a message.

This internal function extracts a Uint32 byte by byte from a given position in the little endian format message. If the Uint32 extends over the size of the message from the given start position ($position + \text{Uint32 byte size} > \text{message size}$), a [radef_kInternalError](#) fatal error message is thrown.

Implements Requirements [RASW-157](#) Endian Definition

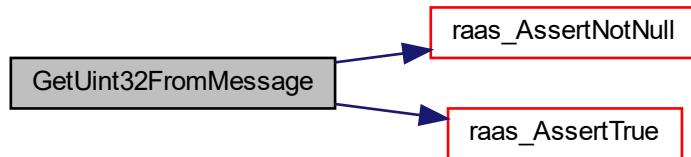
Parameters

in	<i>sr_message</i>	Pointer to a message, from where a variable must be extracted. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	<i>position</i>	Start position where the variable must be read in the message [bytes].

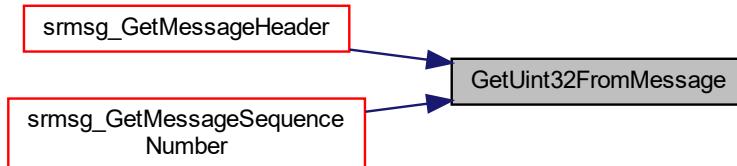
Returns

`uint16_t` Read variable from the specified start position.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.4 IsMessageTypeValid() `static bool IsMessageTypeValid (srtyp_SrMessageType message_type) [static]`

Checks if a message type is valid or not.

This internal function checks the validity of a message type.

Implements Requirements [RASW-616](#) Check Message Function

Parameters

in	<code>message_type</code>	Message type to check. All entries of <code>srtyp_SrMessageType</code> enum are valid. If the value is different from this entries, false is returned.
----	---------------------------	--

Returns

true when message type is valid
 false when message type is invalid/not known

Here is the caller graph for this function:



4.9.2.5 SetMessageHeaderInMessage() static void SetMessageHeaderInMessage (

```

const uint16_t message_length,
const uint16_t message_type,
const srtyp_SrMessageHeaderCreate message_header,
srtyp_SrMessage *const sr_message ) [static]
  
```

Set the message header data in a message.

This internal function sets the header data (message length, message type and [srtyp_SrMessageHeaderCreate](#) consisting of receiver id, sender id and confirmed time stamp) in the provided message. The sequence number, confirmed sequence number and the timestamp are all set to 0, since they are updated just before sending a message.

Implements Requirements [RASW-157](#) Endian Definition

[RASW-160](#) General PDU Message Structure

[RASW-161](#) Message Type

[RASW-162](#) Receiver Identification

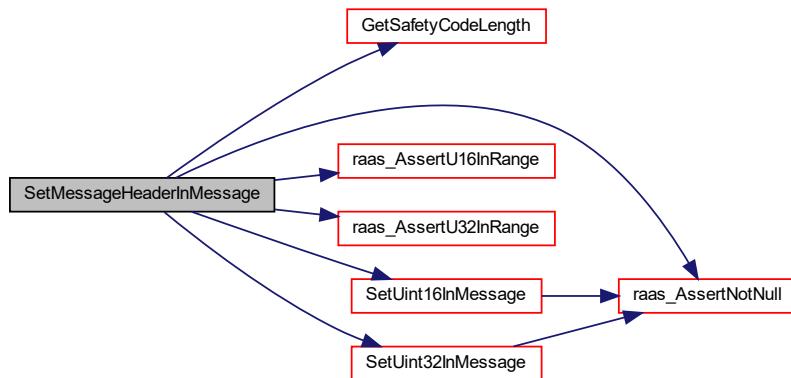
[RASW-163](#) Sender Identification

[RASW-167](#) Confirmed Time Stamp

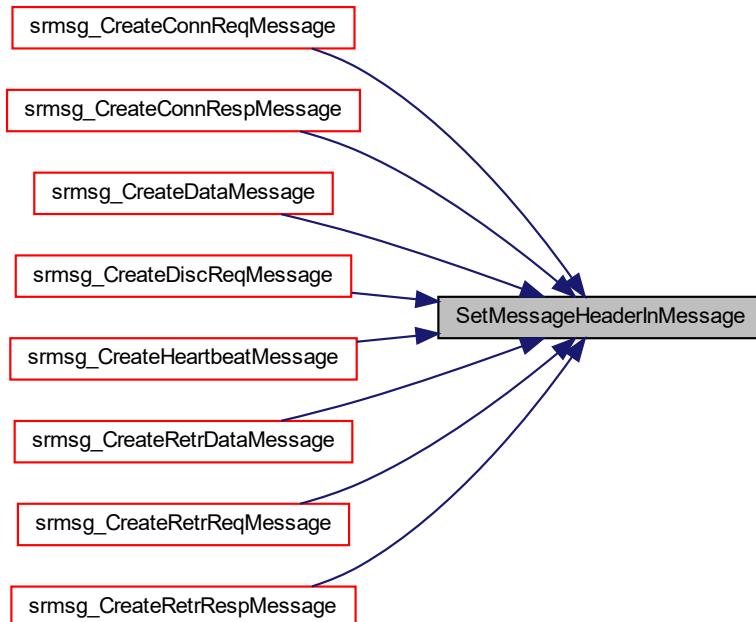
Parameters

in	<i>message_length</i>	Message length to set in message [bytes]. The message length is checked if it is in the valid range of (RADEF_SR_LAYER_MESSAGE_HEADER_SIZE + safety code length) <= value <= RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE , otherwise a radef_kInternalError fatal error is thrown.
in	<i>message_type</i>	Message type to set in message. Valid range: srtyp_kSrMessageMin <= value <= srtyp_kSrMessageMax -1.
in	<i>message_header</i>	Message header data to set in message.
out	<i>sr_message</i>	Pointer to a message, where the header must be set. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.6 SetPayloadDataInMessage() static void SetPayloadDataInMessage (const uint16_t position, const srtyp_SrMessagePayload *const message_payload, srtyp_SrMessage *const sr_message) [static]

Set the payload data in a message.

This internal function write specific payload data at a provided position in the message.

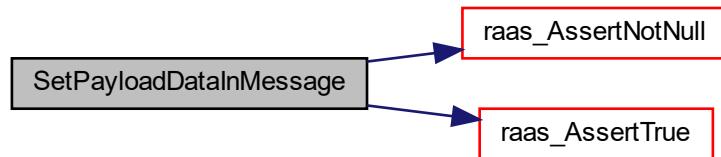
Implements Requirements [RASW-157](#) Endian Definition

[RASW-160](#) General PDU Message Structure

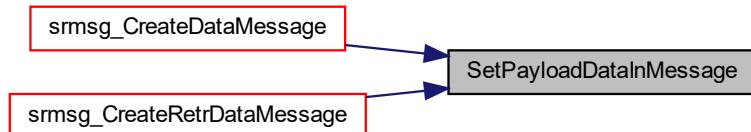
Parameters

in	<i>position</i>	Start position where the payload must be written in the message [bytes].
in	<i>message_payload</i>	Pointer to payload data that is written.
in,out	<i>sr_message</i>	Pointer to a message, where the payload data must be set. If the pointer is NULL, a radef_klInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.7 SetProtocolVersionInMessage() static void SetProtocolVersionInMessage (const srtyp_ProtocolVersion protocol_version, srtyp_SrMessage *const sr_message) [static]

Set the protocol version in a message.

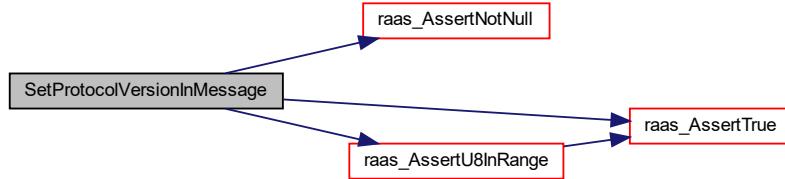
This internal function sets the protocol version in the provided message.

Implements Requirements [RASW-157](#) Endian Definition
[RASW-170](#) Connection Request Message Structure
[RASW-171](#) Connection Response Message Structure
[RASW-173](#) Protocol Version

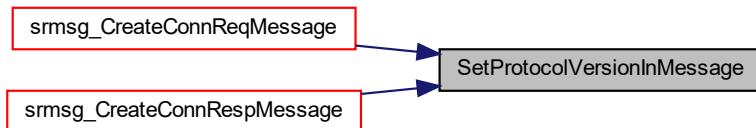
Parameters

in	<i>protocol_version</i>	Protocol version to set in message. Valid range for every digit of the version: <code>srcty_kProtocolVersionMinValue <= value <= srcty_kProtocolVersionMaxValue</code> . If the value is outside this range, a <code>radef_kInternalError</code> fatal error is thrown.
in, out	<i>sr_message</i>	Pointer to a message, where the protocol version must be set. If the pointer is NULL, a <code>radef_kInternalError</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.8 SetUint16InMessage() `static void SetUint16InMessage (const uint16_t position, const uint16_t data, srtyp_SrMessage *const sr_message) [static]`

Set a Uint16 at a specific position in a message.

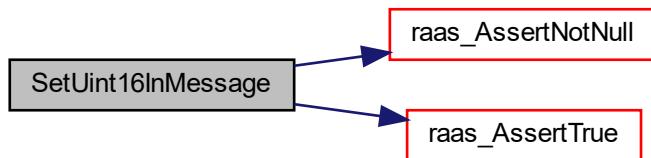
This internal function sets a Uint16 byte by byte in little endian format at a given position in a message. If the Uint16 doesn't fit inside the message (`position + Uint16 byte size > message size`), a `radef_kInternalError` fatal error message is thrown.

Implements Requirements [RASW-157](#) Endian Definition

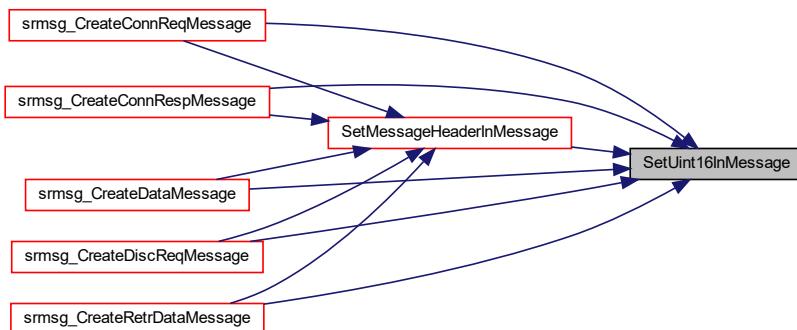
Parameters

<i>in</i>	<i>position</i>	Position where the variable must be set in the message [bytes].
<i>in</i>	<i>data</i>	Variable to be set in the provided message.
<i>in,out</i>	<i>sr_message</i>	Pointer to a message, where the variable must be set. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.9.2.9 SetUint32InMessage() static void SetUint32InMessage (
    const uint16_t position,
    const uint32_t data,
    srtyp_SrMessage *const sr_message ) [static]
  
```

Set a Uint32 at a specific position in a message.

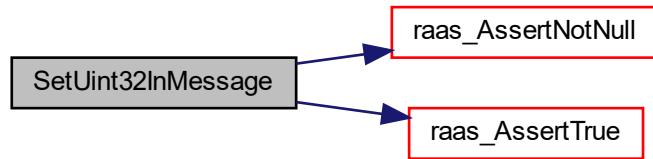
This internal function sets a Uint32 byte by byte in little endian format at a given position in a message. If the Uint32 doesn't fit inside the message (position + Uint32 byte size > message size), a [radef_kInternalError](#) fatal error message is thrown.

Implements Requirements [RASW-157](#) Endian Definition

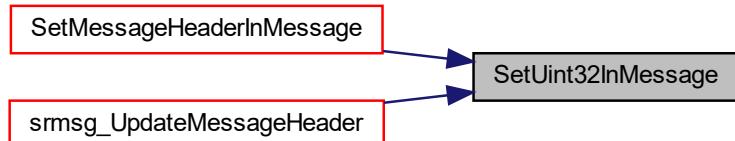
Parameters

in	<i>position</i>	Position where the variable must be set in the message [bytes].
in	<i>data</i>	Variable to be set in the provided message.
in,out	<i>sr_message</i>	Pointer to a message, where the variable must be set. If the pointer is NULL, a raodef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.10 SetUint64InMessage() static void SetUint64InMessage (

```

const uint16_t position,
const uint64_t data,
srtyp_SrMessage *const sr_message ) [static]
  
```

Set a Uint64 at a specific position in a message.

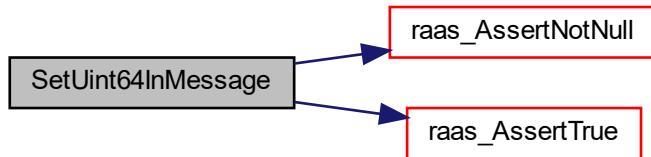
This internal function sets a Uint64 byte by byte in little endian format at a given position in a message. If the Uint64 doesn't fit inside the message (position + Uint64 byte size > message size), a [raodef_kInternalError](#) fatal error message is thrown.

Implements Requirements [RASW-157](#) Endian Definition

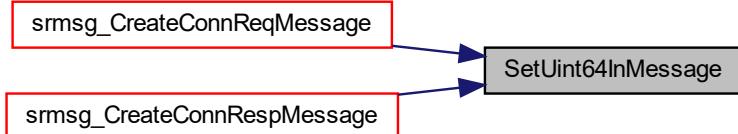
Parameters

in	<i>position</i>	Position where the variable must be set in the message [bytes].
in	<i>data</i>	Variable to be set in the provided message.
in,out	<i>sr_message</i>	Pointer to a message, where the variable must be set. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.11 `srmsg_CheckMessage()` [radef_RaStaReturnCode](#) `srmsg_CheckMessage` (
`const srtyp_SrMessage *const sr_message`)

Check MD4, message type and message size of a SafRetL PDU message.

This function checks the validity of a provided SafRetL message. This means checking the following:

- Safety code (if it is configured). The Safety code inside the message must be identical to the calculated one.
- Message type. The message type must match to one of the known [srtyp_SrMessageType](#) enum entries.
- Message size. The message size must match the expected size following the set message type.
- In case of Data/RetrData: Message payload data must match related to the passed message size (message payload size = message size - [RADEF_SR_LAYER_MESSAGE_HEADER_SIZE](#) - [RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE](#) - safety code length).

All tests must be successful for a message to be valid.

Precondition

The messages module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements `RASW-160` General PDU Message Structure
`RASW-616` Check Message Function

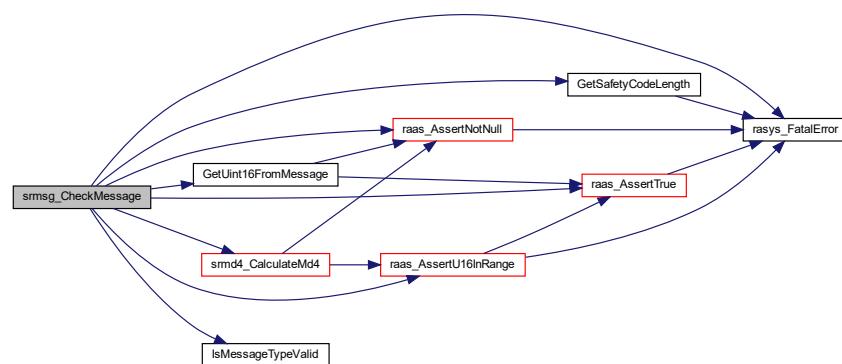
Parameters

in	<code>sr_message</code>	pointer to message to check. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown. The message size is checked if it is in the valid range of $(\text{RADEF_SR_LAYER_MESSAGE_HEADER_SIZE} + \text{safety code length}) \leq \text{value} \leq (\text{RADEF_SR_LAYER_MESSAGE_HEADER_SIZE} + \text{RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE} + \text{RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE} + \text{safety code length})$, otherwise a <code>radef_kInvalidParameter</code> fatal error is thrown.
----	-------------------------	--

Returns

- `radef_kNoError` -> If all test passed and the message is valid.
- `radef_kInvalidMessageMd4` -> If the calculated MD4 doesn't match with the MD4 saved in the message.
- `radef_kInvalidMessageType` -> If the message type is not known or invalid.
- `radef_kInvalidMessageSize` -> If the message size is not correct.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.9.2.12 srmsg_CreateConnReqMessage() void srmsg_CreateConnReqMessage (
    const srtyp\_SrMessageHeaderCreate message_header,
    const srtyp\_ProtocolVersion protocol_version,
    const uint16_t n_send_max,
    srtyp\_SrMessage *const sr_message )
```

Create a new SafRetL connection request message.

This function creates a SafRetL connection request message with the passed header structure ([srtyp_SrMessageHeaderCreate](#) containing sender ID, receiver ID and confirmed_time_stamp), the protocol version [srtyp_ProtocolVersion](#) and its own receive buffer size. The caller must provide a pointer to a memory block, where the function can store the newly created message. The function sets all ConnReq information, including the message length, message type, sender & receiver ID, confirmed time stamp, protocol version and the Nsendmax.

Precondition

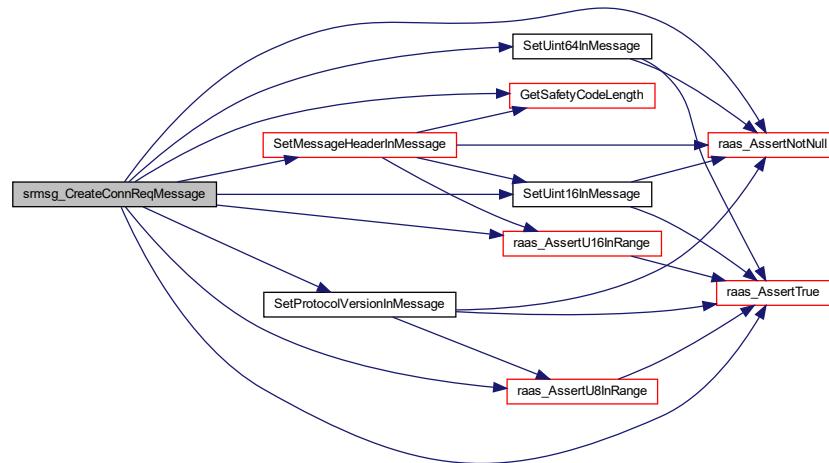
The messages module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-617](#) Create Connection Request Message Function
[RASW-170](#) Connection Request Message Structure
[RASW-172](#) Message Length
[RASW-173](#) Protocol Version
[RASW-174](#) Nsendmax
[RASW-175](#) Reserve

Parameters

in	<i>message_header</i>	Message header data to create a new message. The confirmed time stamp must be 0, otherwise a radef_kInvalidParameter fatal error is thrown. For the other sub-parameters of the header, the full value range is valid and usable.
in	<i>protocol_version</i>	Array containing the protocol version. Valid range for every digit of the version: srcty_kProtocolVersionMinValue <= value <= srcty_kProtocolVersionMaxValue . If the value is outside this range, a radef_kInvalidParameter fatal error is thrown.
in	<i>n_send_max</i>	Nsendmax (receive buffer size). Valid range: srcty_kMinNSendMax <= value <= RADEF_MAX_N_SEND_MAX . If the value is outside this range, a radef_kInvalidParameter fatal error is thrown.
out	<i>sr_message</i>	Pointer to memory block to save the new message. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.13 `srmsg_CreateConnRespMessage()` `void srmsg_CreateConnRespMessage (`
 `const srtyp_SrMessageHeaderCreate message_header,`
 `const srtyp_ProtocolVersion protocol_version,`
 `const uint16_t n_send_max,`
 `srtyp_SrMessage *const sr_message)`

Create a new SafRetL connection response message.

This function creates a SafRetL connection response message with the passed header structure (`srtyp_SrMessageHeaderCreate` containing sender ID, receiver ID and confirmed_time_stamp), the protocol version `srtyp_ProtocolVersion` and its own receive buffer size. The caller must provide a pointer to a memory block, where the function can store the newly created message. The function sets all ConnResp information, including the message length, message type, sender & receiver ID, confirmed time stamp, protocol version and the Nsendmax.

Precondition

The messages module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-618](#) Create Connection Response Message Function

[RASW-171](#) Connection Response Message Structure

[RASW-172](#) Message Length

[RASW-173](#) Protocol Version

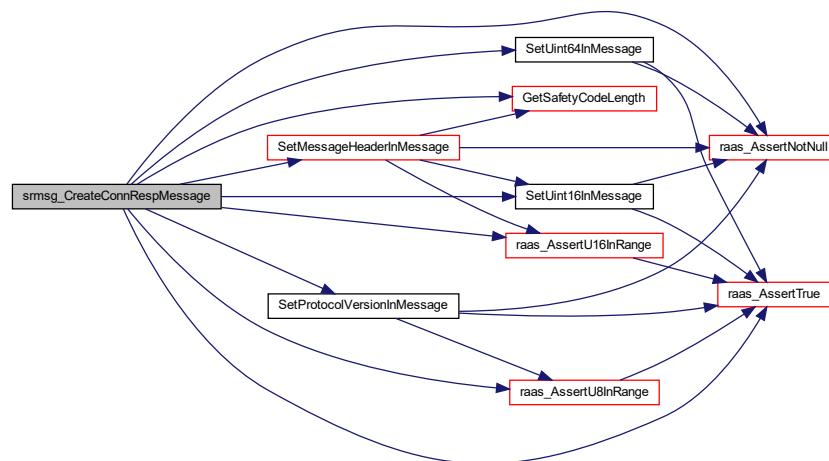
[RASW-174](#) Nsendmax

[RASW-175](#) Reserve

Parameters

in	<i>message_header</i>	Message header data to create a new message. For all sub-parameters of the header, the full value range is valid and usable.
in	<i>protocol_version</i>	Array containing the protocol version. Valid range for every digit of the version: <code>srcty_kProtocolVersionMinValue</code> <= value <= <code>srcty_kProtocolVersionMaxValue</code> . If the value is outside this range, a <code>radef_kInvalidParameter</code> fatal error is thrown.
in	<i>n_send_max</i>	Nsendmax (receive buffer size). Valid range: <code>srcty_kMinNSendMax</code> <= value <= <code>RADEF_MAX_N_SEND_MAX</code> . If the value is outside this range, a <code>radef_kInvalidParameter</code> fatal error is thrown.
out	<i>sr_message</i>	Pointer to memory block to save the new message. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.9.2.14 srmr_CreateDataMessage() void srmr_CreateDataMessage (
    const srtyp_SrMessageHeaderCreate *const message_header,
    const srtyp_SrMessagePayload *const message_payload,
    srtyp_SrMessage *const sr_message )
  
```

Create a new SafRetL data message.

This function creates a SafRetL data message with the passed header structure ([srtyp_SrMessageHeaderCreate](#) containing sender ID, receiver ID and confirmed_time_stamp) and message payload ([srtyp_SrMessagePayload](#) containing payload size and the payload). The caller must provide a pointer to a memory block, where the function can store the newly created message. The function sets all Data information, including the message length, message type, sender & receiver ID, confirmed time stamp, payload length and payload data.

Precondition

The messages module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-623](#) Create Data Message Function

[RASW-191](#) Data Message Structure

[RASW-192](#) Message Length

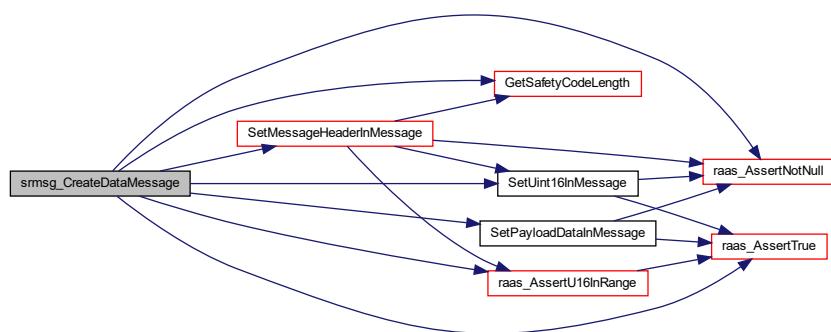
[RASW-193](#) Size of Message Data

[RASW-194](#) Data Message

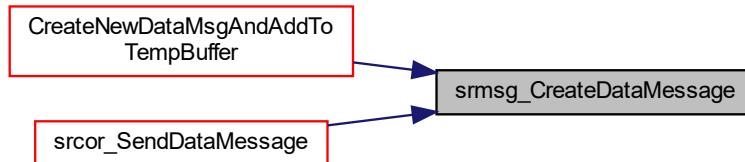
Parameters

in	<i>message_header</i>	Message header data to create a new message. For all sub-parameters of the header, the full value range is valid and usable.
in	<i>message_payload</i>	Pointer to message payload structure. The payload size has a valid range from srcty_kMinSrLayerPayloadDataSize <= value <= RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE , a radef_kInvalidParameter fatal error is thrown otherwise. For the message payload the full value range is valid and usable.
out	<i>sr_message</i>	Pointer to memory block to save the new message. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.15 srmr_CreateDiscReqMessage() void srmr_CreateDiscReqMessage (const [srtyp_SrMessageHeaderCreate](#) message_header, const uint16_t detailed_reason, const [sraty_DiscReason](#) reason, [srtyp_SrMessage](#) *const sr_message)

Create a new SafRetL disconnection request message.

This function creates a SafRetL disconnection request message with the passed header structure ([srtyp_SrMessageHeaderCreate](#) containing sender ID, receiver ID and confirmed_time_stamp), detailed disconnection information and a disconnect reason ([sraty_DiscReason](#)). The caller must provide a pointer to a memory block, where the function can store the newly created message. The function sets all DiscReq information, including the message length, message type, sender & receiver ID, confirmed time stamp, detailed disconnection information and reason.

Precondition

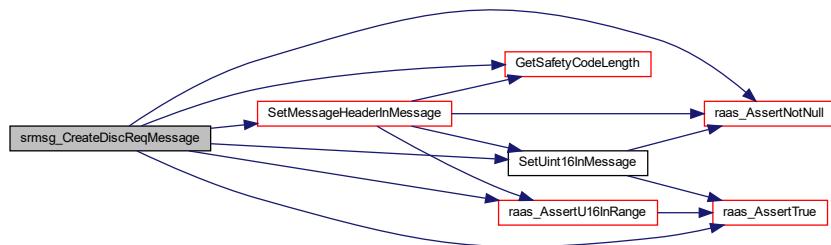
The messages module must be initialized, otherwise a [raodef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-621](#) Create Disconnection Request Message Function
[RASW-183](#) Disconnection Request Message Structure
[RASW-184](#) Message Length
[RASW-185](#) Detailed Information regarding the Reason for the Disconnection Request
[RASW-186](#) Reason for Disconnection Request

Parameters

in	<i>message_header</i>	Message header data to create a new message. For all sub-parameters of the header, the full value range is valid and usable.
in	<i>detailed_reason</i>	Detailed reason for disconnection from application. This is specific to a RaSTA network and can be defined for all parties. The full value range is valid and usable.
in	<i>reason</i>	Reason for disconnection. The reason has a valid range from sraty_kDiscReasonMin <= value < sraty_kDiscReasonMax . If the value is outside this range, a raodef_kInvalidParameter fatal error is thrown.
out	<i>sr_message</i>	Pointer to a memory block to save the new message. If the pointer is NULL, a raodef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.9.2.16 smsg_CreateHeartbeatMessage() void smsg_CreateHeartbeatMessage (
    const srtyp_SrMessageHeaderCreate message_header,
    srtyp_SrMessage *const sr_message )
```

Create a new SafRetL heartbeat message.

This function creates a SafRetL heartbeat message with the passed header structure ([srtyp_SrMessageHeaderCreate](#) containing sender ID, receiver ID and confirmed_time_stamp). The caller must provide a pointer to a memory block, where the function can store the newly created message. The function sets all HB information, including the message length, message type, sender & receiver ID and confirmed time stamp.

Precondition

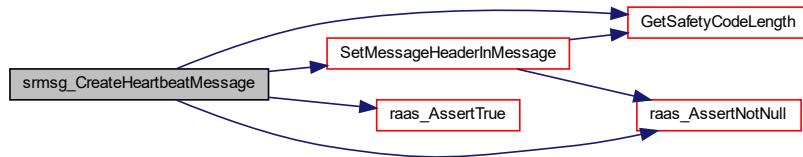
The messages module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-622](#) Create Heartbeat Message Function
[RASW-188](#) Heartbeat Message Structure
[RASW-189](#) Message Length

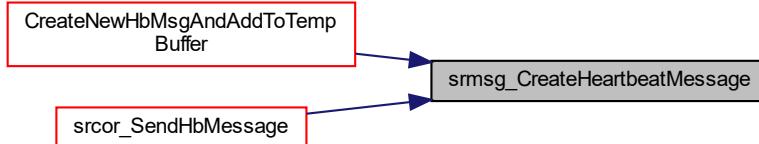
Parameters

in	<i>message_header</i>	Message header data to create a new message. For all sub-parameters of the header, the full value range is valid and usable.
out	<i>sr_message</i>	Pointer to memory block to save the new message. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.9.2.17 srmmsg_CreateRetrDataMessage() void srmmsg_CreateRetrDataMessage (
    const srtyp_SrMessageHeaderCreate *const message_header,
    const srtyp_SrMessagePayload *const message_payload,
    srtyp_SrMessage *const sr_message )
```

Create a new SafRetL retransmitted data message.

This function creates a SafRetL retransmitted data message with the passed header structure (`srtyp_SrMessageHeaderCreate` containing sender ID, receiver ID and confirmed_time_stamp) and message payload (`srtyp_SrMessagePayload` containing payload size and the payload). The caller must provide a pointer to a memory block, where the function can store the newly created message. The function sets all RetrData information, including the message length, message type, sender & receiver ID, confirmed time stamp, payload length and payload data.

Precondition

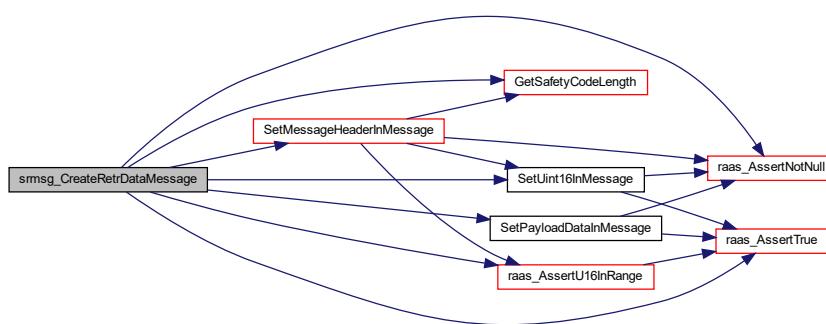
The messages module must be initialized, otherwise a `raodef_kNotInitialized` fatal error is thrown.

Implements Requirements	RASW-624 Create Retransmitted Data Message Function RASW-196 Retransmitted Data Message Structure RASW-192 Message Length RASW-193 Size of Message Data RASW-194 Data Message
--------------------------------	---

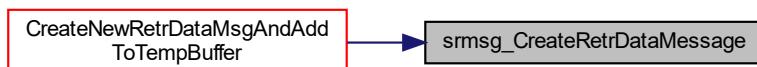
Parameters

in	<i>message_header</i>	Message header data to create a new message. For all sub-parameters of the header, the full value range is valid and usable.
in	<i>message_payload</i>	Pointer to message payload structure. The payload size has a valid range from <code>srcty_kMinSrLayerPayloadDataSize <= value <= RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE</code> , a <code>radef_kInvalidParameter</code> fatal error is thrown otherwise. For the message payload the full value range is valid and usable.
out	<i>sr_message</i>	Pointer to memory block to save the new message. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.9.2.18 srmsg_CreateRetrReqMessage() void srmsg_CreateRetrReqMessage (
    const srtyp_SrMessageHeaderCreate *const message_header,
    srtyp_SrMessage *const sr_message )
  
```

Create a new SafRetL retransmission request message.

This function creates a SafRetL retransmitted request message with the passed header structure (`srtyp_SrMessageHeaderCreate` containing sender ID, receiver ID and confirmed_time_stamp). The caller must provide a pointer to a memory block, where the function can store the newly created message. The function sets all RetrReq information, including the message length, message type, sender & receiver ID and confirmed time stamp.

Precondition

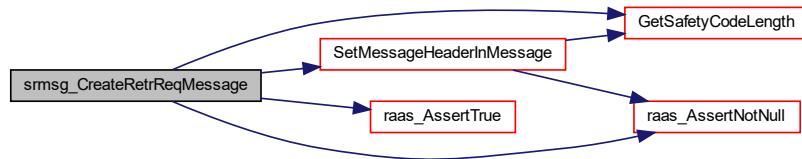
The messages module must be initialized, otherwise a [raodef_kNotInitialized](#) fatal error is thrown.

- Implements Requirements**
- [RASW-619](#) Create Retransmission Request Message Function
 - [RASW-177](#) Retransmission Request Message Structure
 - [RASW-178](#) Message Length

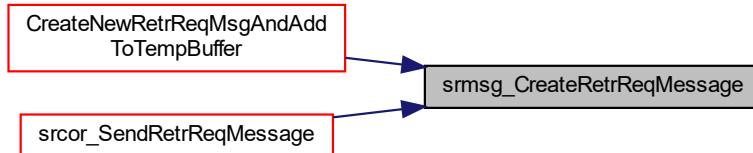
Parameters

in	<i>message_header</i>	Message header data to create a new message. For all sub-parameters of the header, the full value range is valid and usable.
out	<i>sr_message</i>	Pointer to memory block to save the new message. If the pointer is NULL, a raodef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.19 srmsg_CreateRetrReqMessage() void srmsg_CreateRetrReqMessage (

```

const srtyp_SrMessageHeaderCreate message_header,
srtyp_SrMessage *const sr_message )

```

Create a new SafRetL retransmission response message.

This function creates a SafRetL retransmitted response message with the passed header structure ([srtyp_SrMessageHeaderCreate](#) containing sender ID, receiver ID and confirmed_time_stamp). The caller must provide a pointer to a memory block, where the function can store the newly created message. The function sets all RetrReq information, including the message length, message type, sender & receiver ID and confirmed time stamp.

Precondition

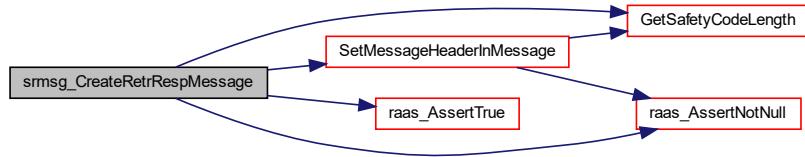
The messages module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

- Implements Requirements**
- [RASW-620](#) Create Retransmission Response Message Function
 - [RASW-180](#) Retransmission Response Message Structure
 - [RASW-178](#) Message Length

Parameters

in	<i>message_header</i>	Message header data to create a new message. For all sub-parameters of the header, the full value range is valid and usable.
out	<i>sr_message</i>	Pointer to memory block to save the new message. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.20 srmsg_GetConnMessageData() void srmsg_GetConnMessageData (

```

const srtyp_SrMessage *const sr_message,
srtyp_ProtocolVersion *const protocol_version,
uint16_t *const n_send_max )
  
```

Get the data of a SafRetL connection request or connection response message.

This function extracts the connection message data ([srtyp_ProtocolVersion](#) and receive buffer size from opposite side) from a passed [srtyp_kSrMessageConnReq](#) or [srtyp_kSrMessageConnResp](#) message.

Precondition

The messages module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-625](#) Get Connection Message Data Function

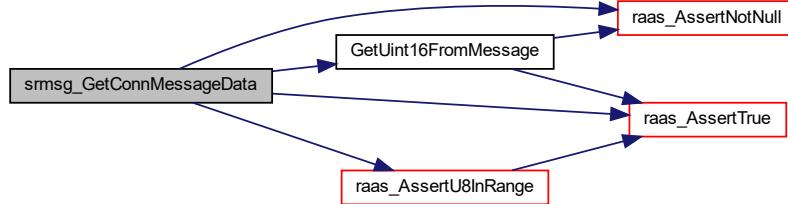
[RASW-173](#) Protocol Version

[RASW-174](#) Nsendmax

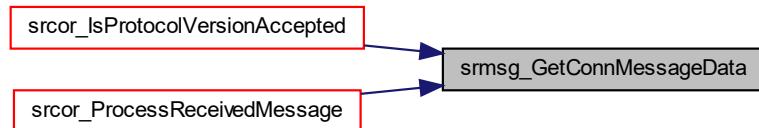
Parameters

in	<i>sr_message</i>	Pointer to a memory block containing a message. If the pointer is NULL or the message is an other message type then srtyp_kSrMessageConnReq or srtyp_kSrMessageConnResp , a radef_kInvalidParameter fatal error is thrown. Every digit of the protocol version should be in the valid ASCII range of srcty_kProtocolVersionMinValue <= value <= srcty_kProtocolVersionMaxValue , otherwise a radef_kInvalidParameter fatal error is thrown.
out	<i>protocol_version</i>	Pointer to memory block containing srtyp_ProtocolVersion structure. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
out	<i>n_send_max</i>	Pointer to a memory block containing Nsendmax (receive buffer size). If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.9.2.21 srmsg_GetDataMessagePayload() void srmsg_GetDataMessagePayload (
    const srtyp_SrMessage *const sr_message,
    srtyp_SrMessagePayload *const message_payload )
```

Get the payload of a SafRetL data or retransmitted data message.

This function extracts the message payload size & data from a passed `srtyp_kSrMessageData` or `srtyp_kSrMessageRetrData` message. If the message contains more data than allowed (`RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE`), a `radef_kInvalidParameter` fatal error is thrown.

Precondition

The messages module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-626](#) Get Data Message Payload Function

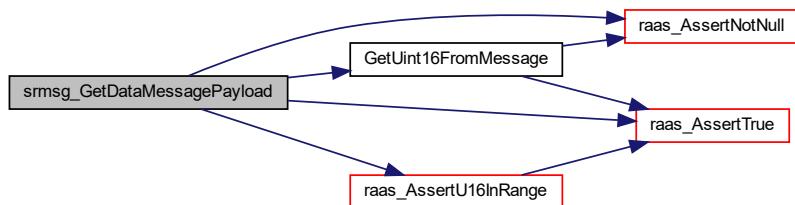
[RASW-193](#) Size of Message Data

[RASW-194](#) Data Message

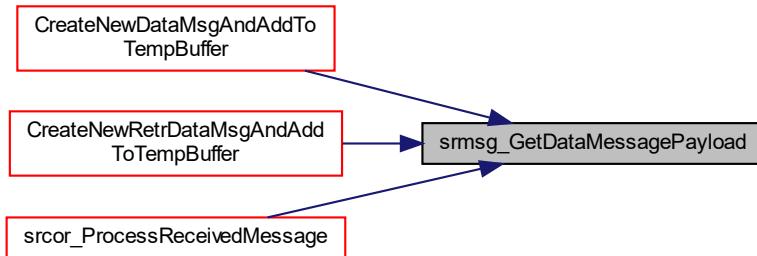
Parameters

in	<i>sr_message</i>	Pointer to a memory block containing a message. If the pointer is NULL or the message is an other message type then <code>srtyp_kSrMessageData</code> or <code>srtyp_kSrMessageRetrData</code> , a <code>radef_kInvalidParameter</code> fatal error is thrown. For the message payload size, the following range is valid: <code>srcty_kMinSrLayerPayloadDataSize <= value <= RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE</code> .
out	<i>message_payload</i>	Pointer to memory block for storing the message payload. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.22 srmmsg_GetDiscMessageData() void srmmsg_GetDiscMessageData (const [srtyp_SrMessage](#) *const *sr_message*, uint16_t *const *detailed_reason*, [sraty_DiscReason](#) *const *reason*)

Get the data of a SafRetL disconnection request message.

This function extracts the disconnection request message data (detailed reason and [sraty_DiscReason](#)) from a passed [srtyp_KSrMessageDiscReq](#) message. If the message is an other message type, a [raodef_kInvalidParameter](#) fatal error is thrown. When the extracted reason is not in the valid range of [sraty_kDiscReasonMin](#) <= value < [sraty_kDiscReasonMax](#), a [raodef_kInvalidParameter](#) fatal error is thrown.

Precondition

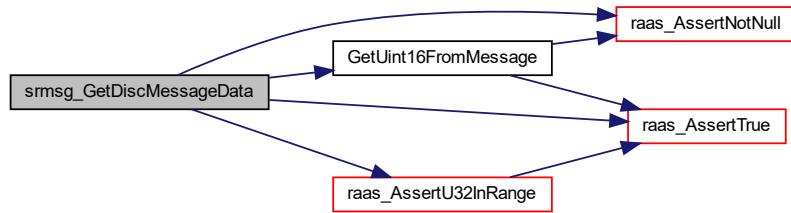
The messages module must be initialized, otherwise a [raodef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-627](#) Get Disconnection Message Data Function
[RASW-185](#) Detailed Information regarding the Reason for the Disconnection Request
[RASW-186](#) Reason for Disconnection Request

Parameters

in	<i>sr_message</i>	Pointer to a memory block containing a message. If the pointer is NULL or the message is an other message type then srtyp_KSrMessageDiscReq , a raodef_kInvalidParameter fatal error is thrown.
out	<i>detailed_reason</i>	Pointer to a memory block for storing the detailed reason for disconnection from application. If the pointer is NULL, a raodef_kInvalidParameter fatal error is thrown.
out	<i>reason</i>	Pointer to a memory block for storing the sraty_DiscReason . If the pointer is NULL, a raodef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



4.9.2.23 `srmsg_GetMessageHeader()` `void srmsg_GetMessageHeader (`
`const srtyp_SrMessage *const sr_message,`
`srtyp_SrMessageHeader *const message_header)`

Get the header of a SafRetL PDU message.

This function extracts the header data (`srtyp_SrMessageHeader` containing the message length, message type, receiver & sender ID, sequence number, confirmed sequence number, time stamp and confirmed time stamp) from the passed SafRetL message.

Precondition

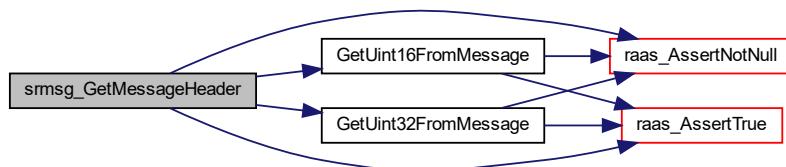
The messages module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-628](#) Get Message Header Function
[RASW-160](#) General PDU Message Structure
[RASW-161](#) Message Type
[RASW-162](#) Receiver Identification
[RASW-163](#) Sender Identification
[RASW-164](#) Sequence Number
[RASW-165](#) Confirmed Sequence Number
[RASW-166](#) Time Stamp
[RASW-167](#) Confirmed Time Stamp

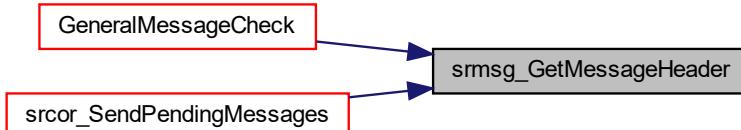
Parameters

in	<code>sr_message</code>	Pointer to a memory block containing a message. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.
out	<code>message_header</code>	Pointer to memory block for storing header data. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.24 `srmsg_GetMessageSequenceNumber()` `uint32_t srmsg_GetMessageSequenceNumber (`
`const srtyp_SrMessage *const sr_message)`

Get the sequence number of a SafRetL PDU message.

This function extracts and returns the sequence number from the passed SafRetL message.

Precondition

The messages module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-825](#) Get Message Sequence Number Function

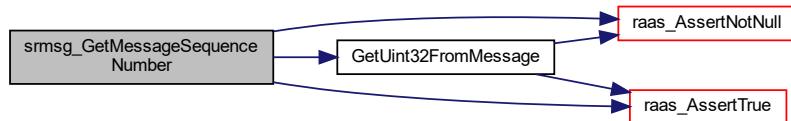
Parameters

<code>sr_message</code>	Pointer to a memory block containing a message. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.
-------------------------	---

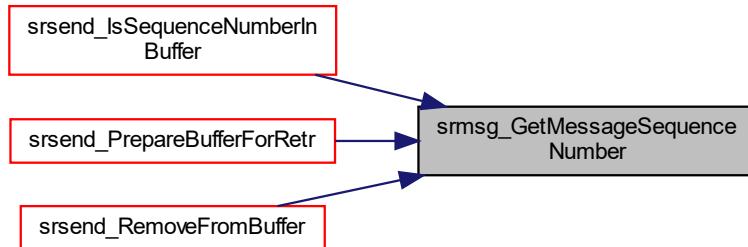
Returns

`uint32_t` Sequence number from the passed message

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.25 `srmsg_GetMessageType()` `srtyp_SrMessageType srmsg_GetMessageType (const srtyp_SrMessage *const sr_message)`

Get the message type of a SafRetL PDU message.

This function extracts and returns the message type from the passed SafRetL message.

Precondition

The messages module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-824](#) Get Message Type Function

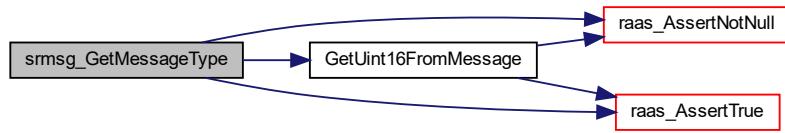
Parameters

<code>sr_message</code>	Pointer to a memory block containing a message. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown.
-------------------------	---

Returns

`srtyp_SrMessageType` Message type from the passed message

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.26 `smsg_Init()` `void smsg_Init (`
 `const srcty_SafetyCodeType configured_safety_code_type,`
 `const srcty_Md4InitValue configured_md4_initial_value)`

Initialize SafRetL messages module.

This function is used to initialize the messages module. It saves the passed safety code type and the MD4 initial value. A fatal error is raised, if this function is called multiple times.

Precondition

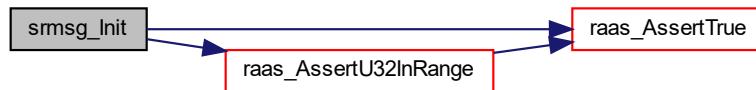
The messages module must not be initialized, otherwise a `radef_kAlreadyInitialized` fatal error is thrown.

Implements Requirements [RASW-629](#) Init sr_messages Function

Parameters

in	<code>configured_safety_code_type</code>	Configured RaSTA safety code type. All enum entries of <code>srcty_SafetyCodeType</code> are valid and usable.
in	<code>configured_md4_initial_value</code>	Configured MD4 initial value. The full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.27 srmsg_UpdateMessageHeader() `void srmsg_UpdateMessageHeader (`
`const srtyp_SrMessageHeaderUpdate message_header_update,`
`srtyp_SrMessage *const sr_message)`

Update a SafRetL message header and calculate the safety code to prepare the message for sending.

This function updates the header data (`srtyp_SrMessageHeaderUpdate` containing the sequence number, confirmed sequence number and timestamp) in the provided SafRetL message and then calculates and sets the safety code, if it is configured.

Precondition

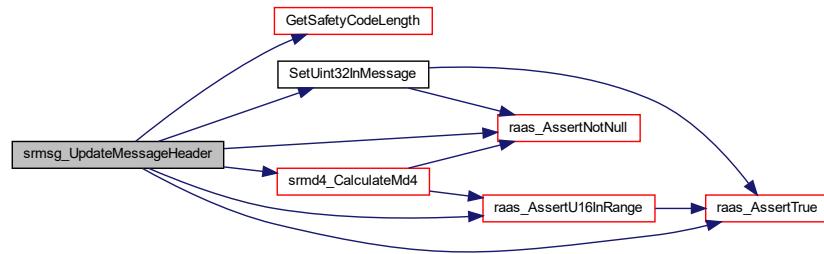
The messages module must be initialized, otherwise a `radef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-630](#) Update Message Header Function
[RASW-164](#) Sequence Number
[RASW-165](#) Confirmed Sequence Number
[RASW-166](#) Time Stamp
[RASW-168](#) Safety Code

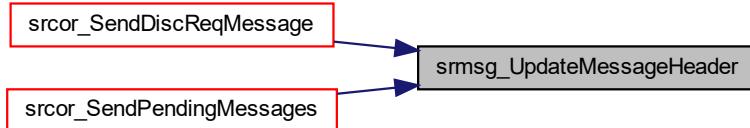
Parameters

in	<code>message_header_update</code>	Message header data to update. For all sub-parameters of the <code>srtyp_SrMessageHeaderUpdate</code> structure, the full value range is valid and usable.
in,out	<code>sr_message</code>	Pointer to a memory block to save new message. If the pointer is NULL, a <code>radef_kInvalidParameter</code> fatal error is thrown. The message size is checked if it is in the valid range of $(\text{RADEF_SR_LAYER_MESSAGE_HEADER_SIZE} + \text{safety code length}) \leq \text{value} \leq \text{RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE}$, otherwise a <code>radef_kOutOfRangeParameter</code> fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10 MD4 component

Interface of RaSTA SafRetL MD4 module.

Collaboration diagram for MD4 component:



Classes

- struct [srmd4_Md4](#)
Typedef for MD4 result.

Functions

- static void `Md4Body` (`Md4Context` *const ctx, const `uint8_t` *const data, const `uint32_t` number_of_blocks)
Inner function for MD4 calculation.
- static void `Md4Update` (`Md4Context` *const ctx, const `uint8_t` *const data, const `uint32_t` size)
Calculation of MD4. Allows incremental calculation.
- static void `Md4Final` (`Md4Context` *ctx, `uint8_t` *result)
Final MD4 calculations, generating the MD4 hash.
- static void `SetContextBuffer` (const `uint8_t` value, const `uint32_t` size, const `uint32_t` md4_context_buffer_start_index, `Md4Context` *const md4_context)
Helper function to set the MD4 context data buffer memory to a dedicated value.
- static void `CopyToContextBuffer` (const `uint8_t` *const source, const `uint32_t` size, const `uint32_t` md4_context_buffer_start_index, `Md4Context` *const md4_context)
Helper function to copy memory to a MD4 context data buffer memory.
- static void `ClearMd4ContextData` (`Md4Context` *const md4_context)
Helper function to clear the MD4 context data.
- static void `WriteU32toByteArray` (const `uint32_t` source, `uint8_t` *const destination)
Helper to write a Uint32 value in a byte array.
- static `uint32_t` `FunctionF` (const `uint32_t` x, const `uint32_t` y, const `uint32_t` z)
Basic MD4 calculation function F, which returns $F = z \wedge (x \wedge y \wedge z)$.
- static `uint32_t` `FunctionG` (const `uint32_t` x, const `uint32_t` y, const `uint32_t` z)
Basic MD4 calculation function G, which returns $G = (x \wedge (y \mid z)) \mid (y \wedge z)$.
- static `uint32_t` `FunctionH` (const `uint32_t` x, const `uint32_t` y, const `uint32_t` z)
Basic MD4 calculation function H, which returns $H = x \wedge y \wedge z$.
- static void `Step` (const `Md4Function` md4_function, `uint32_t` *const a, const `uint32_t` b, const `uint32_t` c, const `uint32_t` d, const `uint32_t` x, const `uint32_t` s)
The MD4 transformation function for all three rounds.
- static `uint32_t` `Set` (const `uint8_t` index_32_bit, const `uint8_t` *const data, `Md4Context` *const ctx)
Reads 4 input data bytes and stores them in a properly aligned Uint32 value in a MD4 context data block and returns this Uint32 value.
- static `uint32_t` `Get` (const `uint8_t` index_32_bit, const `Md4Context` *const ctx)
Get a properly aligned Uint32 value from a MD4 context data block.
- void `srmd4_CalculateMd4` (const `srcty_Md4InitValue` md4_initial_value, const `uint16_t` data_size, const `uint8_t` *const data_buffer, `srmd4_Md4` *const calculated_md4)
Calculate the MD4 of a data buffer.

4.10.1 Detailed Description

Interface of RaSTA SafRetL MD4 module.

Specification of the MD4 component of the RaSTA Safety and Retransmission Layer.

This module provides the MD4 hash calculation functionality as specified in RFC 1320.

This implementation is based on the "A portable, fast, and free implementation of the MD4 Message-Digest Algorithm (RFC 1320)" implementation of Alexander Peslyak in 2001. www.openwall.info/wiki/people/solar/software/public-domain-source-code/md4

The source code is refactored to achieve the requirements of MISRA C:2012 and EN50128 for SIL4.

Implements Requirements

[RASW-632](#) Component sr_md4 Overview

[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

[RASW-520](#) Error Handling

[RASW-521](#) Input Parameter Check

4.10.2 Function Documentation

4.10.2.1 ClearMd4ContextData() static void ClearMd4ContextData (
`Md4Context *const md4_context) [static]`

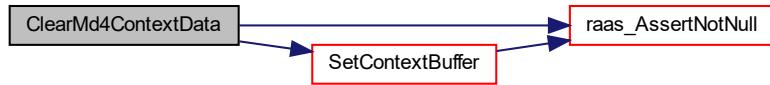
Helper function to clear the MD4 context data.

Implements Requirements RASW-633 Calculate MD4 Function

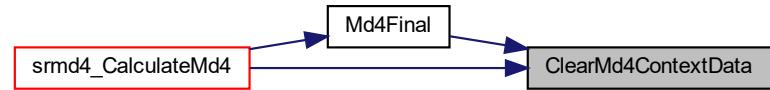
Parameters

in, out	<code>md4_context</code>	The MD4 context. If the pointer is NULL, a <code>radef_kInternalError</code> fatal error is thrown.
---------	--------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.2.2 CopyToContextBuffer() static void CopyToContextBuffer (
`const uint8_t *const source,`
`const uint32_t size,`
`const uint32_t md4_context_buffer_start_index,`
`Md4Context *const md4_context) [static]`

Helper function to copy memory to a MD4 context data buffer memory.

Precondition

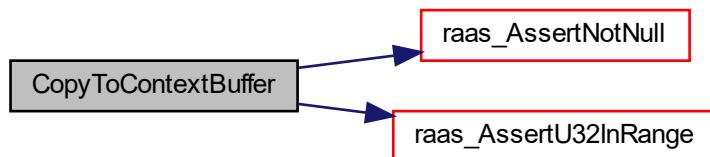
Following condition must be respected: size + md4_context_buffer_start_index has a valid range of: 0U <= value <=MD4_INPUT_DATA_BLOCK_SIZE.

Implements Requirements [RASW-633](#) Calculate MD4 Function

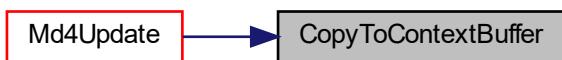
Parameters

in	<i>source</i>	Pointer to source data buffer.
in	<i>size</i>	Size of data to copy [bytes]. Valid range: $0 \leq \text{value} \leq \text{MD4_INPUT_DATA_BLOCK_SIZE}$.
in	<i>md4_context_buffer_start_index</i>	Write start index in the MD4 context buffer [bytes]. Valid range: $0 \leq \text{value} < \text{MD4_INPUT_DATA_BLOCK_SIZE}$.
in,out	<i>md4_context</i>	The MD4 context. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.2.3 FunctionF() static uint32_t FunctionF (const uint32_t x, const uint32_t y, const uint32_t z) [static]

Basic MD4 calculation function F, which returns $F = z \wedge (x \wedge (y \wedge z))$.

Implements Requirements [RASW-633](#) Calculate MD4 Function

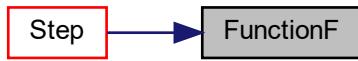
Parameters

in	<i>x</i>	Input parameter x. The full value range is valid and usable.
in	<i>y</i>	Input parameter y. The full value range is valid and usable.
in	<i>z</i>	Input parameter z. The full value range is valid and usable.

Returns

unit32_t The calculation result.

Here is the caller graph for this function:



4.10.2.4 FunctionG() static uint32_t FunctionG (const uint32_t x, const uint32_t y, const uint32_t z) [static]

Basic MD4 calculation function G, which returns $G = (x \& (y | z)) | (y \& z)$.

Implements Requirements RASW-633 Calculate MD4 Function

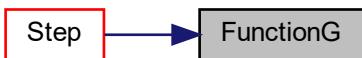
Parameters

in	x	Input parameter x. The full value range is valid and usable.
in	y	Input parameter y. The full value range is valid and usable.
in	z	Input parameter z. The full value range is valid and usable.

Returns

unit32_t The calculation result.

Here is the caller graph for this function:



```
4.10.2.5 FunctionH() static uint32_t FunctionH (
    const uint32_t x,
    const uint32_t y,
    const uint32_t z ) [static]
```

Basic MD4 calculation function H, which returns $H = x \wedge y \wedge z$.

Implements Requirements RASW-633 Calculate MD4 Function

Parameters

in	x	Input parameter x. The full value range is valid and usable.
in	y	Input parameter y. The full value range is valid and usable.
in	z	Input parameter z. The full value range is valid and usable.

Returns

unit32_t The calculation result.

Here is the caller graph for this function:



```
4.10.2.6 Get() static uint32_t Get (
    const uint8_t index_32_bit,
    const Md4Context *const ctx ) [static]
```

Get a properly aligned Uint32 value from a MD4 context data block.

Implements Requirements RASW-633 Calculate MD4 Function

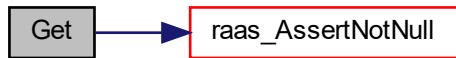
Parameters

in	index_32_bit	Index in the MD4 context data block [bytes]. The full value range is valid and usable.
in	ctx	Pointer to the MD4 context data structure. If the pointer is NULL, a raef_KInternalError fatal error is thrown.

Returns

`unit32_t` Requested `Uint32` value at given index.

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.10.2.7 Md4Body() static void Md4Body (
    Md4Context *const ctx,
    const uint8_t *const data,
    const uint32_t number_of_blocks ) [static]
    
```

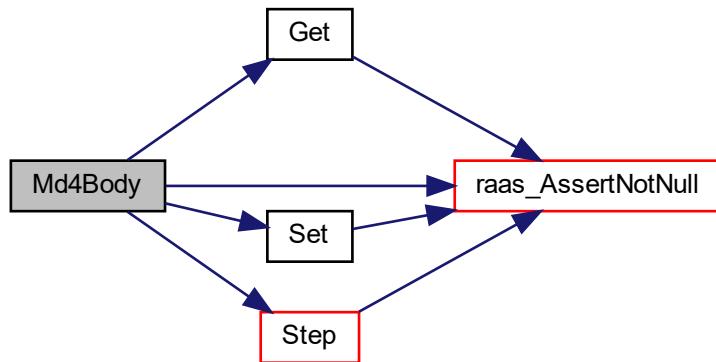
Inner function for MD4 calculation.

Implements Requirements [RASW-633](#) Calculate MD4 Function
[RASW-634](#) Safety Code

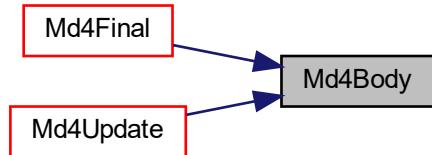
Parameters

<code>in, out</code>	<code>ctx</code>	Pointer to the MD4 context data structure. If the pointer is NULL, a radef_KInternalError fatal error is thrown.
<code>in</code>	<code>data</code>	Pointer to the input data. If the pointer is NULL, a radef_KInternalError fatal error is thrown.
<code>in</code>	<code>number_of_blocks</code>	Number of the 64 byte input data blocks [blocks]. Full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.2.8 Md4Final() static void Md4Final (
 `Md4Context * ctx,`
 `uint8_t * result) [static]`

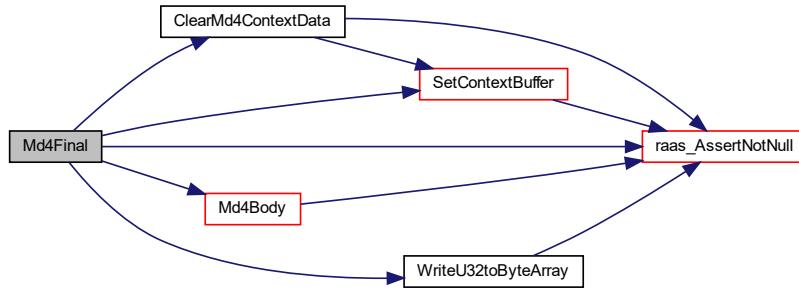
Final MD4 calculations, generating the MD4 hash.

Implements Requirements [RASW-633](#) Calculate MD4 Function
[RASW-634](#) Safety Code

Parameters

in, out	<code>ctx</code>	Pointer to the MD4 context data structure. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
out	<code>result</code>	The MD4 hash result. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.10.2.9 Md4Update() static void Md4Update (
    Md4Context *const ctx,
    const uint8_t *const data,
    const uint32_t size ) [static]

```

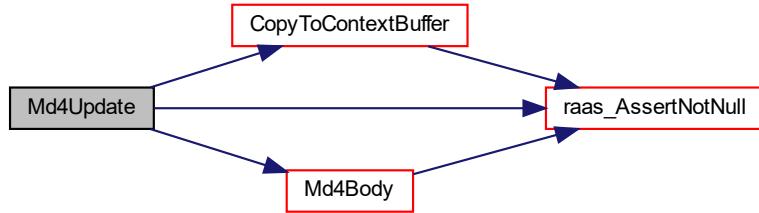
Calculation of MD4. Allows incremental calculation.

Implements Requirements [RASW-633](#) Calculate MD4 Function
[RASW-634](#) Safety Code

Parameters

in, out	<i>ctx</i>	Pointer to the MD4 context data structure. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	<i>data</i>	Pointer to the input data. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	<i>size</i>	Size of the input data [bytes]. Full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.10.2.10 Set() static uint32_t Set (
    const uint8_t index_32_bit,
    const uint8_t *const data,
    Md4Context *const ctx ) [static]
```

Reads 4 input data bytes and stores them in a properly aligned Uint32 value in a MD4 context data block and returns this Uint32 value.

Implements Requirements RASW-633 Calculate MD4 Function

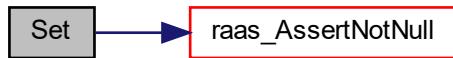
Parameters

in	<i>index_32_bit</i>	Index in the MD4 context data block [bytes]. The full value range is valid and usable.
in	<i>data</i>	Pointer to start of current input data block. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in, out	<i>ctx</i>	Pointer to the MD4 context data structure. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Returns

`uint32_t` Set Uint32 value at given index.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.2.11 SetContextBuffer() `static void SetContextBuffer (`
 `const uint8_t value,`
 `const uint32_t size,`
 `const uint32_t md4_context_buffer_start_index,`
 `Md4Context *const md4_context) [static]`

Helper function to set the MD4 context data buffer memory to a dedicated value.

Precondition

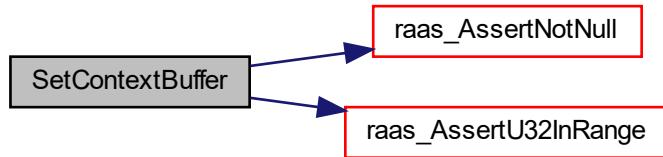
Following condition must be respected: `size + md4_context_buffer_start_index` has a valid range of: `0U <= value <= MD4_INPUT_DATA_BLOCK_SIZE`.

Implements Requirements [RASW-633](#) Calculate MD4 Function

Parameters

<code>in</code>	<code>value</code>	Value to set. Full value range is valid and usable.
<code>in</code>	<code>size</code>	Size of data to copy [bytes]. Valid range: <code>0 <= value <= MD4_INPUT_DATA_BLOCK_SIZE</code> .
<code>in</code>	<code>md4_context_buffer_start_index</code>	Write start index in the MD4 context buffer [bytes]. Valid range: <code>0 <= value < MD4_INPUT_DATA_BLOCK_SIZE</code> .
<code>in, out</code>	<code>md4_context</code>	The MD4 context. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.2.12 `srm4_CalculateMd4()` `void srm4_CalculateMd4 (`
`const srcty_Md4InitValue md4_initial_value,`
`const uint16_t data_size,`
`const uint8_t *const data_buffer,`
`srm4_Md4 *const calculated_md4)`

Calculate the MD4 of a data buffer.

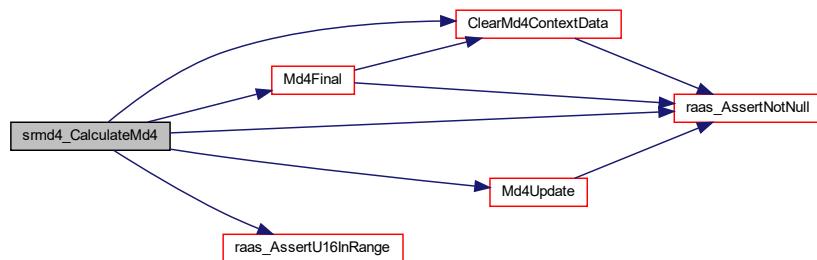
Implements Requirements [RASW-633](#) Calculate MD4 Function

[RASW-634](#) Safety Code

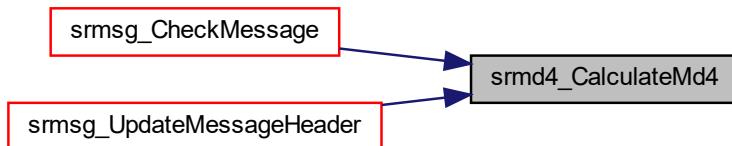
Parameters

in	<code>md4_initial_value</code>	MD4 initial value. Valid values are defined in srcty_Md4InitValue .
in	<code>data_size</code>	size of data buffer [bytes]. Valid range: <code>RADEF_SR_LAYER_MESSAGE_HEADER_SIZE <= value <= (RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE - RADEF_MAX_SR_LAYER_SAFETY_CODE_SIZE)</code> .
in	<code>data_buffer</code>	Pointer to data buffer. If the pointer is NULL, a radef_klInvalidParameter fatal error is thrown.
out	<code>calculated_md4</code>	Pointer to array with calculated MD4. If the pointer is NULL, a radef_klInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



```

4.10.2.13 Step() static void Step (
    const Md4Function md4_function,
    uint32_t *const a,
    const uint32_t b,
    const uint32_t c,
    const uint32_t d,
    const uint32_t x,
    const uint32_t s ) [static]
  
```

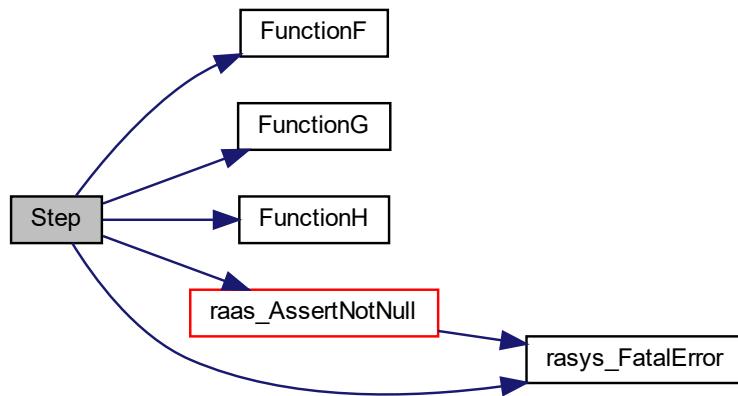
The MD4 transformation function for all three rounds.

Implements Requirements RASW-633 Calculate MD4 Function

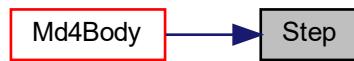
Parameters

in	md4_function	Basic MD4 function used for this step. The values <code>kMd4FunctionF</code> , <code>kMd4FunctionG</code> , <code>kMd4FunctionH</code> are valid and usable. For other values a <code>radef_kInternalError</code> fatal error is thrown.
in,out	a	Input/output parameter a. The full value range is valid and usable.
in	b	Input parameter b. The full value range is valid and usable.
in	c	Input parameter c. The full value range is valid and usable.
in	d	Input parameter d. The full value range is valid and usable.
in	x	Input parameter x. The full value range is valid and usable.
in	s	Input parameter s. The full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.2.14 WriteU32toByteArray() static void WriteU32toByteArray (const uint32_t source, uint8_t *const destination) [static]

Helper to write a Uint32 value in a byte array.

Implements Requirements RASW-633 Calculate MD4 Function

Parameters

in	source	Uint32 source value. Full value range is valid and usable.
out	destination	Pointer to destination byte array. If the pointer is NULL, a radef_kInternalError fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11 Diagnostics component

Interface of RaSTA SafRetL diagnostics module.

Collaboration diagram for Diagnostics component:



Classes

- struct [srdia_SrConnectionDiagnostics](#)
Struct for the collection of diagnostic data of a RaSTA connection.

Functions

- static void [AddTimeToTimingDistribution](#) (uint32_t *const distribution_array, const uint32_t time_value)
Increment the matching distribution interval for a new time in a specific distribution array.
- bool [srdia_AreDiagnosticTimingIntervalsValid](#) (const uint32_t t_max, const uint32_t *const diag_timing_distr_intervals)
Checks if the diagnostic timings distribution intervals are valid.

- void [srdia_Init](#) (const uint32_t configured_connections, const uint32_t t_max, const uint32_t n_diag_window, const uint32_t *const diag_timing_distr_intervals)
Initialize SafRetL diagnostics module.
- void [srdia_InitConnectionDiagnostics](#) (const uint32_t connection_id)
Initialize the diagnostic data of a dedicated RaSTA connection.
- void [srdia_IncSafetyCodeErrorCounter](#) (const uint32_t connection_id)
Increment the safety code error counter of a dedicated RaSTA connection.
- void [srdia_IncAddressErrorCounter](#) (const uint32_t connection_id)
Increment the address error counter of a dedicated RaSTA connection.
- void [srdia_IncTypeErrorHandler](#) (const uint32_t connection_id)
Increment the message type error counter of a dedicated RaSTA connection.
- void [srdia_IncSequenceNumberErrorCounter](#) (const uint32_t connection_id)
Increment the sequence number error counter of a dedicated RaSTA connection.
- void [srdia_IncConfirmedSequenceNumberErrorCounter](#) (const uint32_t connection_id)
Increment the confirmed sequence number error counter of a dedicated RaSTA connection.
- void [srdia_UpdateConnectionDiagnostics](#) (const uint32_t connection_id, const uint32_t round_trip_delay, const uint32_t alive_time)
Update the received message timing statistics of a dedicated RaSTA connection.
- void [srdia_SendDiagnosticNotification](#) (const uint32_t connection_id)
Send a diagnostic notification of a dedicated RaSTA connection.

4.11.1 Detailed Description

Interface of RaSTA SafRetL diagnostics module.

Specification of the Diagnostics component of the RaSTA Safety and Retransmission Layer.

This module provides all functionalities for the SafRetL diagnostics. It handles all possible error counters and the connection diagnostics. There are error counters for messages received with:

- faulty safety code
- implausible sender or receiver identification
- undefined message type
- implausible sequence number
- implausible confirmed sequence number

The connection diagnostics contain information about the distribution of messages inside the defined diagnostic window. Information is collected for the round trip time and the alive time.

Implements Requirements [RASW-636](#) Component sr_diagnostics Overview

[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

[RASW-520](#) Error Handling

[RASW-521](#) Input Parameter Check

4.11.2 Function Documentation

```
4.11.2.1 AddTimeToTimingDistribution() static void AddTimeToTimingDistribution (
    uint32_t *const distribution_array,
    const uint32_t time_value ) [static]
```

Increment the matching distribution interval for a new time in a specific distribution array.

This function evaluates in which time interval a new time belongs and increments the matching distribution interval. T_max is splitted into 5 intervals which are defined in [srdia_diag_timing_distr_intervals](#). The following bins are defined:

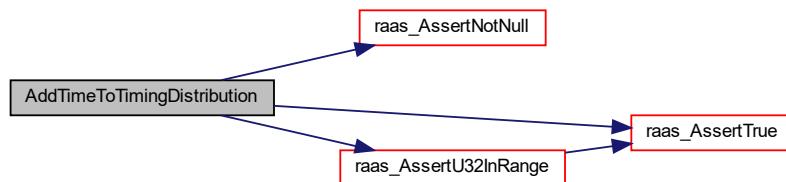
- bin 1: $0 \leq value \leq T1$
- bin 2: $T1 < value \leq T2$
- bin 3: $T2 < value \leq T3$
- bin 4: $T3 < value \leq T4$
- bin 5: $T4 < value \leq t_{max}$

Implements Requirements [RASW-645](#) Update Connection Diagnostics Function
[RASW-433](#) Diagnostic Timing Interval

Parameters

in	<i>distribution_array</i>	Pointer to distribution array in which the matching distribution interval is incremented. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	<i>time_value</i>	Time value to be added to diagnostic distribution. Valid range: $0 \leq value \leq$ configured t_max.

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.11.2.2 srdia_AreDiagnosticTimingIntervalsValid() bool srdia_AreDiagnosticTimingIntervalsValid (
    const uint32_t t_max,
    const uint32_t *const diag_timing_distr_intervals )
```

Checks if the diagnostic timings distribution intervals are valid.

Check the validity of the diagnostic timing distribution intervals. Detailed description for this check can be found in [srcty_SafetyRetransmissionConfiguration::diag_timing_distr_intervals](#).

Implements Requirements [RASW-817](#) Are Diagnostic Timing Intervals Valid Function

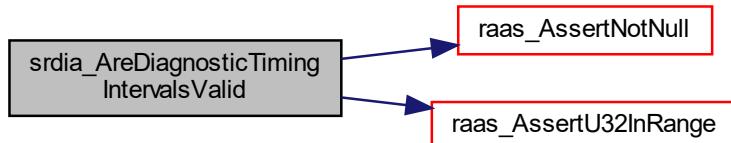
Parameters

in	<i>t_max</i>	Configured maximal accepted age of a message [ms]. Valid range: srcty_kMinTMax <= value <= srcty_kMaxTMax .
in	<i>diag_timing_distr_intervals</i>	Configured diagnostic timing distribution intervals [ms]. For every element full value range is valid and usable. They will be checked by this function for their validity. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

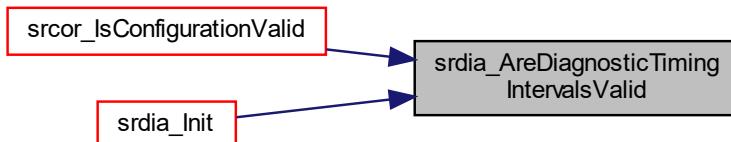
Returns

true, if the intervals are valid
false, if the intervals are invalid

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.3 srdia_IncAddressErrorCounter() void srdia_IncAddressErrorCounter (const uint32_t connection_id)

Increment the address error counter of a dedicated RaSTA connection.

This function is used to increment the error counter for messages received with an implausible sender or receiver identification of a dedicated connection.

Precondition

The diagnostics module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-637](#) Inc Address Error Counter Function

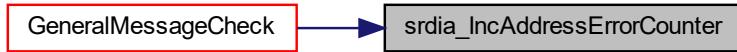
Parameters

in	connection_id	RaSTA connection identification. Valid range: 0 <= value < configured number of connections.
----	---------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.4 srdia_IncConfirmedSequenceNumberErrorCounter() void srdia_IncConfirmedSequenceNumberErrorCounter (const uint32_t connection_id)

Increment the confirmed sequence number error counter of a dedicated RaSTA connection.

This function is used to increment the error counter for messages received with an implausible confirmed sequence number of a dedicated connection.

Precondition

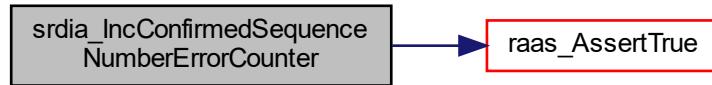
The diagnostics module must be initialized, otherwise a [raodef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-638](#) Inc Confirmed Sequence Number Error Counter Function

Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.5 srdia_IncSafetyCodeErrorCounter() `void srdia_IncSafetyCodeErrorCounter (const uint32_t connection_id)`

Increment the safety code error counter of a dedicated RaSTA connection.

This function is used to increment the error counter for messages received with a faulty safety code of a dedicated connection.

Precondition

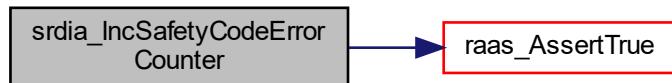
The diagnostics module must be initialized, otherwise a [raodef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-639](#) Inc Safety Code Error Counter Function

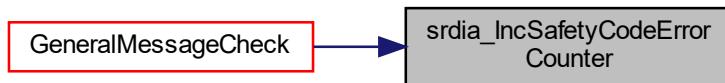
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.6 srdia_IncSequenceNumberErrorCounter() `void srdia_IncSequenceNumberErrorCounter (const uint32_t connection_id)`

Increment the sequence number error counter of a dedicated RaSTA connection.

This function is used to increment the error counter for messages received with an implausible sequence number of a dedicated connection.

Precondition

The diagnostics module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-640](#) Inc Sequence Number Error Counter Function

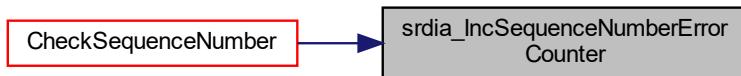
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.7 srdia_IncTypeErrorHandler() `void srdia_IncTypeErrorHandler (const uint32_t connection_id)`

Increment the message type error counter of a dedicated RaSTA connection.

This function is used to increment the error counter for messages received with undefined message type of a dedicated connection.

Precondition

The diagnostics module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-641](#) Inc Type Error Counter Function

Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.8 srdia_Init()

```

void srdia_Init (
    const uint32_t configured_connections,
    const uint32_t t_max,
    const uint32_t n_diag_window,
    const uint32_t *const diag_timing_distr_intervals )
  
```

Initialize SafRetL diagnostics module.

This function is used to initialize the diagnostic data of all connections. It saves the passed maximal age of a message (*t_max*), diagnostic window size (*n_diag_window*) and the diagnostic timing distribution intervals. Additionally, [srdia_InitConnectionDiagnostics](#) is called for all configured connections.

Precondition

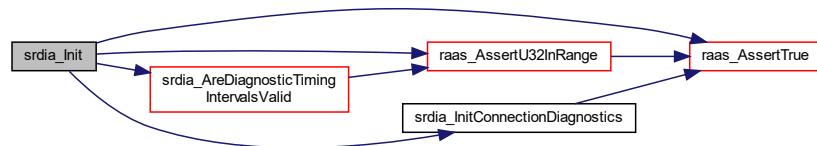
The diagnostics module must not be initialized, otherwise a [radef_kAlreadyInitialized](#) fatal error is thrown.

Implements Requirements [RASW-642](#) Init sr_diagnostics Function

Parameters

in	<i>configured_connections</i>	Number of configured RaSTA connections. Valid range: $1 \leq \text{value} \leq \text{RADEF_MAX_NUMBER_OF_RSTA_CONNECTIONS}$.
in	<i>t_max</i>	Configured maximal accepted age of a message [ms]. Valid range: $\text{srcty_kMinTMax} \leq \text{value} \leq \text{srcty_kMaxTMax}$.
in	<i>n_diag_window</i>	Configured diagnosis window size [messages]. Valid range: $\text{srcty_kMinNDiagWindow} \leq \text{value} \leq \text{srcty_kMaxNDiagWindow}$.
in	<i>diag_timing_distr_intervals</i>	Configured diagnostic timing distribution intervals [ms].

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.9 `srdia_InitConnectionDiagnostics()` `void srdia_InitConnectionDiagnostics (const uint32_t connection_id)`

Initialize the diagnostic data of a dedicated RaSTA connection.

This function is used to initialize the diagnostic data of a dedicated connection. This resets the message counter of `srdia_SrConnectionDiagnostics` and also all members in the embedded `sraty_ConnectionDiagnosticData` structure including all error counters and the distribution arrays.

Precondition

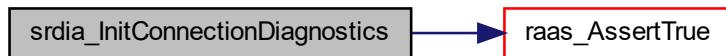
The diagnostics module must be initialized, otherwise a `raodef_kNotInitialized` fatal error is thrown.

Implements Requirements [RASW-643](#) Init Connection Diagnostics Function

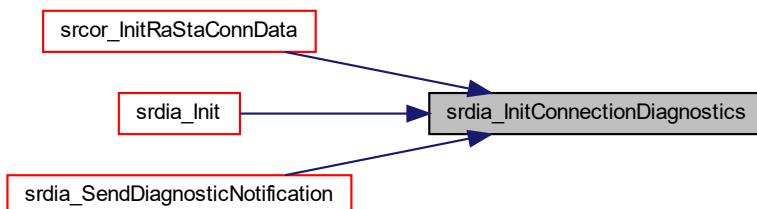
Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} < \text{configured number of connections}$.
----	----------------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.10 srdia_SendDiagnosticNotification() `void srdia_SendDiagnosticNotification (const uint32_t connection_id)`

Send a diagnostic notification of a dedicated RaSTA connection.

This function is used to send a diagnostic notification ([srnot_SrDiagnosticNotification](#)) to the application layer containing the current diagnostic data of a dedicated RaSTA connection. After sending the notification, the diagnostic data of the dedicated connection is reset using [srdia_InitConnectionDiagnostics](#).

Precondition

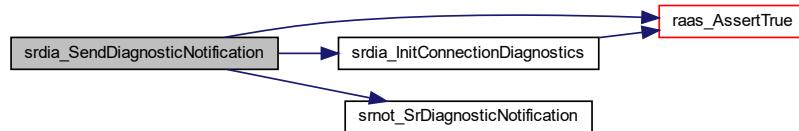
The diagnostics module must be initialized, otherwise a [radef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-644](#) Send Diagnostic Notification Function

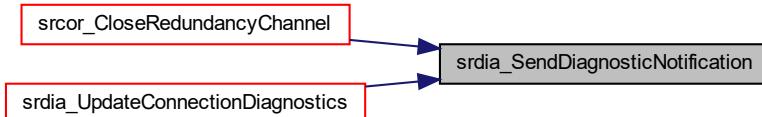
Parameters

in	<i>connection_id</i>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
----	----------------------	---

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.11 `srdia_UpdateConnectionDiagnostics()`

```

void srdia_UpdateConnectionDiagnostics (
    const uint32_t connection_id,
    const uint32_t round_trip_delay,
    const uint32_t alive_time )
  
```

Update the received message timing statistics of a dedicated RaSTA connection.

This function is used to update the received message timing statistics for a newly received time-out related message. For the passed round trip delay and alive time, the corresponding timing distributions are updated according the passed value by calling [AddTimeToTimingDistribution](#). Additionally, the internal received messages counter is incremented and if the diagnosis window (`n_diag_window`) is reached (message counter $\geq n_diag_window$), a diagnostic notification is send with the current diagnostics data ([srdia_SendDiagnosticNotification](#)).

Precondition

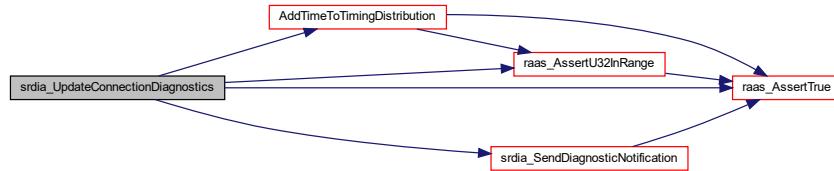
The diagnostics module must be initialized, otherwise a [raodef_kNotInitialized](#) fatal error is thrown.

Implements Requirements [RASW-645](#) Update Connection Diagnostics Function

Parameters

in	<code>connection_id</code>	RaSTA connection identification. Valid range: $0 \leq \text{value} <$ configured number of connections.
in	<code>round_trip_delay</code>	Round trip delay of a received message (<code>T_rtd</code>) [ms]. Valid range: $0 \leq \text{value} \leq$ configured <code>t_max</code> .
in	<code>alive_time</code>	Alive time of a received message (<code>T_alive</code>) [ms]. Valid range: $0 \leq \text{value} \leq$ configured <code>t_max</code> .

Here is the call graph for this function:



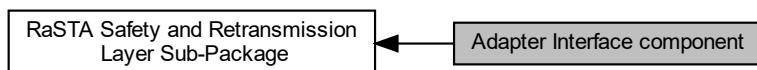
Here is the caller graph for this function:



4.12 Adapter Interface component

Interface of RaSTA SafRetL adapter.

Collaboration diagram for Adapter Interface component:



Functions

- void `sradin_Init` (void)
Initialize SafRetL adapter.
- void `sradin_OpenRedundancyChannel` (const uint32_t redundancy_channel_id)
Open a redundancy channel.
- void `sradin_CloseRedundancyChannel` (const uint32_t redundancy_channel_id)
Close a redundancy channel.
- void `sradin_SendMessage` (const uint32_t redundancy_channel_id, const uint16_t message_size, const uint8_t *const message_data)
Send a message over a redundancy channel.
- `radef_RaStaReturnCode sradin_ReadMessage` (const uint32_t redundancy_channel_id, const uint16_t buffer_size, uint16_t *const message_size, uint8_t *const message_buffer)
Read a received message from a redundancy channel.

4.12.1 Detailed Description

Interface of RaSTA SafRetL adapter.

Specification of the Adapter Interface component of the RaSTA Safety and Retransmission Layer.

This module defines the interface functions (like init, open & close redundancy channel, send & read message) for the SafRetL adapter interface. The SafRetL only defines the interface, the implementation of this adapter interface functions must be done in the SafRetL adapter.

Remarks

The error handling for all function must be implemented and handled by the system integrator when developing the SafRetL adapter.

Implements Requirements [RASW-647](#) Component sr_adapter_interface Overview

[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

4.12.2 Function Documentation

4.12.2.1 sradin_CloseRedundancyChannel() `void sradin_CloseRedundancyChannel (`
`const uint32_t redundancy_channel_id)`

Close a redundancy channel.

This function is used to close a specific redundancy channel of the RedL. The SafRetL only defines the interface function, the implementation of this interface function must be done in the SafRetL adapter.

Remarks

There is a 1:1 mapping between the connection id and redundancy channel id.

Implements Requirements [RASW-650](#) Close Redundancy Channel Function

[RASW-368](#) Close Redundancy Channel Function Structure

[RASW-367](#) Redundancy Channel Id

Parameters

in	<code>redundancy_channel_id</code>	Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.
----	------------------------------------	--

Here is the caller graph for this function:



4.12.2.2 sradin_Init() void sradin_Init (void)

Initialize SafRetL adapter.

This function is used to initialize the sr_adapter_interface module. The SafRetL only defines the interface function, the implementation of this interface function must be done in the SafRetL adapter.

Implements Requirements [RASW-648](#) Init sr_adapter_interface Function
[RASW-353](#) Initialization Function Structure

4.12.2.3 sradin_OpenRedundancyChannel() void sradin_OpenRedundancyChannel (const uint32_t redundancy_channel_id)

Open a redundancy channel.

This function is used to open a specific redundancy channel of the RedL. The SafRetL only defines the interface function, the implementation of this interface function must be done in the SafRetL adapter.

Remarks

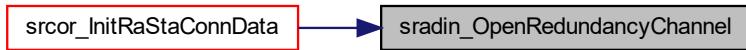
There is a 1:1 mapping between the connection id and redundancy channel id.

Implements Requirements [RASW-649](#) Open Redundancy Channel Function
[RASW-369](#) Open Redundancy Channel Function Structure
[RASW-367](#) Redundancy Channel Id

Parameters

in	<i>redundancy_channel_id</i>	Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.
----	------------------------------	--

Here is the caller graph for this function:



4.12.2.4 `sradin_ReadMessage()`

```

radef_RaStaReturnCode sradin_ReadMessage (
    const uint32_t redundancy_channel_id,
    const uint16_t buffer_size,
    uint16_t *const message_size,
    uint8_t *const message_buffer )

```

Read a received message from a redundancy channel.

This function is used to read a SafRetL message from a specific redundancy channel of the RedL. The SafRetL only defines the interface function, the implementation of this interface function must be done in the SafRetL adapter.

Implements Requirements [RASW-652](#) Read Message Function

[RASW-375](#) Read Message Function Structure
[RASW-381](#) Redundancy Channel Id
[RASW-379](#) Buffer Size
[RASW-372](#) Message Size
[RASW-371](#) Message Buffer
[RASW-374](#) Error Code
[RASW-503](#) Enum RaSta Return Code Usage

Parameters

in	<i>redundancy_channel_id</i>	Redundancy channel identification. Valid range: $0 \leq \text{value} <$ configured number of redundancy channels.
in	<i>buffer_size</i>	Size of the buffer provided to parameter <i>message_buffer</i> [bytes]. Valid range: RADEF_SR_LAYER_MESSAGE_HEADER_SIZE $\leq \text{value} \leq$ RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE . Any value in this range can be used, must be large enough to store the received message.
out	<i>message_size</i>	Pointer to the size of the received message data [bytes].
out	<i>message_buffer</i>	Pointer to a buffer for saving the received message. Enough memory to save a message with <i>buffer_size</i> must be allocated.

Returns

`radef_kNoError` -> successful operation
`radef_kNoMessageReceived` -> no message received (used for polling)

Here is the caller graph for this function:



4.12.2.5 sradin_SendMessage() void sradin_SendMessage (

- const uint32_t redundancy_channel_id,
- const uint16_t message_size,
- const uint8_t *const message_data)

Send a message over a redundancy channel.

This function is used to send a SafRetL message over a specific redundancy channel of the RedL. The SafRetL only defines the interface function, the implementation of this interface function must be done in the SafRetL adapter.

Remarks

There is a 1:1 mapping between the connection id and redundancy channel id.

Implements Requirements [RASW-651](#) Send Message Function

[RASW-364](#) Send Message Function Structure

[RASW-363](#) Redundancy Channel Id

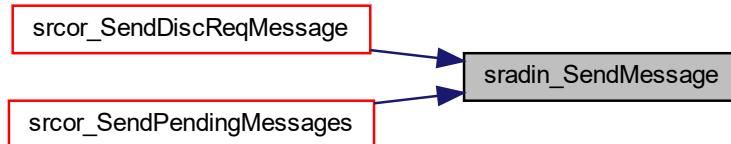
[RASW-387](#) Message Size

[RASW-385](#) Message Data

Parameters

in	<i>redundancy_channel_id</i>	Redundancy channel identification. Valid range: $0 \leq \text{value} < \text{configured number of redundancy channels}$.
in	<i>message_size</i>	Size of the message data [bytes]. Valid range: RADEF_SR_LAYER_MESSAGE_HEADER_SIZE $\leq \text{value} \leq$ RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE .
in	<i>message_data</i>	Pointer to message data array. For the message data the full value range is valid and usable.

Here is the caller graph for this function:



4.13 Adapter Notifications component

Interface of RaSTA SafRetL adapter notifications.

Collaboration diagram for Adapter Notifications component:



Functions

- `raodef_RaStaReturnCode sradno_MessageReceivedNotification (const uint32_t red_channel_id)`
Message received notification from the RedL.
- `raodef_RaStaReturnCode sradno_DiagnosticNotification (const uint32_t red_channel_id, const uint32_t tr_channel_id, const raodef_TransportChannelDiagnosticData tr_channel_diagnostic_data)`
Diagnostic notification from the RedL.

4.13.1 Detailed Description

Interface of RaSTA SafRetL adapter notifications.

Specification of the Adapter Notifications component of the RaSTA Safety and Retransmission Layer.

This module defines and implements the interface notifications (like message received notification and diagnostic notification) for the SafRetL, as they act as entry point to for the SafRetL adapter.

- Implements Requirements**
- [RASW-654 Component sr_adapter_notifications Overview](#)
 - [RASW-518 Safety and Retransmission Layer Safety Integrity Level](#)
 - [RASW-520 Error Handling](#)
 - [RASW-521 Input Parameter Check](#)

4.13.2 Function Documentation

4.13.2.1 sradno_DiagnosticNotification() `radef_RaStaReturnCode sradno_DiagnosticNotification (`

```
const uint32_t red_channel_id,
const uint32_t tr_channel_id,
const radef_TransportChannelDiagnosticData tr_channel_diagnostic_data )
```

Diagnostic notification from the RedL.

This function is called by the SafRetL adapter to notify the SafRetL about new diagnostic data from the RedL. The redundancy channel id is checked if it is in a valid range, otherwise a `radef_kInvalidParameter` is returned. The passed diagnostic data contains the transport channel identification, diagnostic window size, number of missed messages and the average delay indicators Tdrift und Tdrift2.

Implements Requirements [RASW-656](#) Diagnostic Notification

[RASW-332](#) Diagnostic Notification Structure

[RASW-331](#) Redundancy Channel Id

[RASW-334](#) Transport Channel Id

[RASW-333](#) Transport Channel Diagnostic Data

[RASW-901](#) Error Code

[RASW-503](#) Enum RaSta Return Code Usage

[RASW-751](#) Redundancy Diagnostic Notification Sequence

Parameters

in	<code>red_channel_id</code>	Redundancy channel identification. Valid range: $0 \leq \text{value} < \text{RADEF_MAX_NUMBER_OF_RSTA_CONNECTIONS}$.
in	<code>tr_channel_id</code>	Transport channel identification. The full value range is valid and usable.
in	<code>tr_channel_diagnostic_data</code>	Transport channel diagnostic data. Structure and valid ranges can be found in <code>radef_TransportChannelDiagnosticData</code> .

Returns

`radef_kNoError` -> successful operation

`radef_kInvalidParameter` -> invalid parameter

Here is the call graph for this function:



```
4.13.2.2 sradno_MessageReceivedNotification() raodef_RaStaReturnCode sradno_MessageReceived→
Notification (
    const uint32_t red_channel_id )
```

Message received notification from the RedL.

This function is called by the SafRetL adapter to notify the SafRetL that a received message from a specific redundancy channel is ready to be read. The redundancy channel id is checked if it is in a valid range, otherwise a `raodef_kInvalidParameter` is returned. New messages are read as long as received messages pending and enough space in send and received buffers (checked with `srcor_IsReceivedMsgPendingAndBuffersNotFull`).

Remarks

There is a 1:1 mapping between the connection id and redundancy channel id.

Implements Requirements

- [RASW-655](#) Message Received Notification
- [RASW-335](#) Message Received Notification Structure
- [RASW-338](#) Redundancy Channel Id
- [RASW-900](#) Error Code
- [RASW-503](#) Enum RaSta Return Code Usage
- [RASW-747](#) Message Received Notification Sequence
- [RASW-769](#) Received Message Polling Sequence

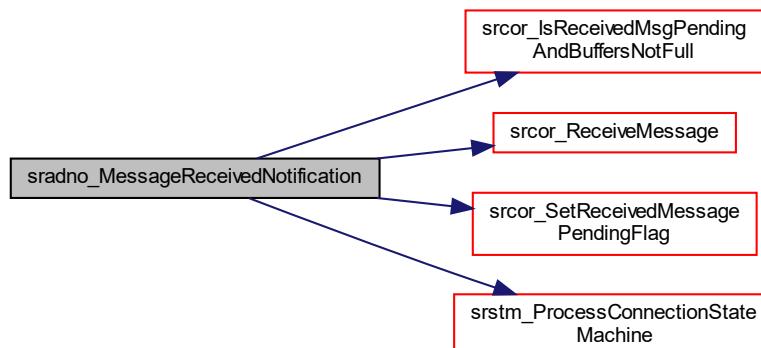
Parameters

in	<code>red_channel_id</code>	Redundancy channel identification. Valid range: $0 \leq \text{value} < \text{RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS}$.
----	-----------------------------	--

Returns

`raodef_kNoError` -> successful operation
`raodef_kInvalidParameter` -> invalid parameter

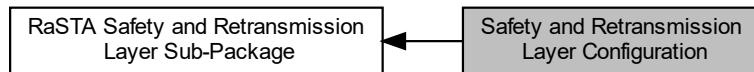
Here is the call graph for this function:



4.14 Safety and Retransmission Layer Configuration

Type definitions of RaSTA SafRetL configuration.

Collaboration diagram for Safety and Retransmission Layer Configuration:



Classes

- struct `srcty_ConnectionConfiguration`
Struct for the configuration of a RaSTA connection.
- struct `srcty_Md4InitValue`
Struct for the MD4 initial value for a RaSTA network.
- struct `srcty_SafetyRetransmissionConfiguration`
Struct for the configuration data of the SafRetL.

Enumerations

- enum `srcty_SafetyCodeType` {

 `srcty_kSafetyCodeTypeMin` = 1 , `srcty_kSafetyCodeTypeNone` = 1 , `srcty_kSafetyCodeTypeLowerMd4` = 2 ,

 `srcty_kSafetyCodeTypeFullMd4` = 3 ,

 `srcty_kSafetyCodeTypeMax` }

Enum for the safety code type.

Variables

- const `uint32_t srcty_kMinNumberOfRaStaConnections`
Minimum number of RaSTA connections per RaSTA network.
- const `uint16_t srcty_kMinSrLayerPayloadDataSize`
Minimum SafRetL payload data size.
- const `uint32_t srcty_kMinTMax`
Minimum max. accepted age of a message (T_{max}) [ms].
- const `uint32_t srcty_kMaxTMax`
Maximum max. accepted age of a message (T_{max}) [ms].
- const `uint32_t srcty_kMinTHeartbeat`
Minimum heartbeat period (T_h) [ms].
- const `uint32_t srcty_kMaxTHeartbeat`
Maximum heartbeat period (T_h) [ms].
- const `uint16_t srcty_kMinNSendMax`
Minimum receive buffer size ($N_{sendmax}$) [messages].
- const `uint16_t srcty_kMinMWA`
Minimum max. number of received, unconfirmed messages (MWA) [messages].

- const uint16_t **srcty_kMaxMWA**
Maximum max. number of received, unconfirmed messages (MWA) [messages].
- const uint32_t **srcty_kNMaxPacket**
Packetization factor (must always be 1!).
- const uint32_t **srcty_kMinNDiagWindow**
Minimum SafRetL diagnosis window size (Ndiagwindow) [messages].
- const uint32_t **srcty_kMaxNDiagWindow**
Maximum SafRetL diagnosis window size (Ndiagwindow) [messages].
- const uint16_t **srcty_kByteCountUint16**
Byte count of type UInt16_t [bytes].
- const uint16_t **srcty_kByteCountUint32**
Byte count of type UInt32_t [bytes].
- const uint16_t **srcty_kByteCountUint64**
Byte count of type UInt64_t [bytes].
- const uint8_t **srcty_kProtocolVersionMinValue**
Minimum ASCII character value for protocol verion.
- const uint8_t **srcty_kProtocolVersionMaxValue**
Maximum ASCII character value for protocol verion.
- const uint32_t **srcty_kMinFreeEntriesSendBufferForRetr**
Minimum amount of free entries in the send buffer in case of a retransmission.
- const uint32_t **srcty_kMinFreeEntriesReceivedBufferForReceive**
Minimum amount of free entries in the received buffer in case of receiving a message.

4.14.1 Detailed Description

Type definitions of RaSTA SafRetL configuration.

Specification of the Configuration of the RaSTA Safety and Retransmission Layer.

This module defines the data types and data structures used for the RaSTA SafRetL configuration.

Implements Requirements [RASW-518](#) Safety and Retransmission Layer Safety Integrity Level
[RASW-487](#) Enum Safety Code Type Structure
[RASW-423](#) Struct Connection Configuration Structure
[RASW-437](#) Struct MD4 Initial Value Structure
[RASW-427](#) Struct SafetyRetransmissionConfiguration Structure

4.14.2 Enumeration Type Documentation

4.14.2.1 **srcty_SafetyCodeType** enum [srcty_SafetyCodeType](#)

Enum for the safety code type.

Implements Requirements [RASW-487](#) Enum Safety Code Type Structure

Enumerator

<code>srcty_kSafetyCodeTypeMin</code>	Min value for safety code enum.
<code>srcty_kSafetyCodeTypeNone</code>	Safety code type 1: No safety code.
<code>srcty_kSafetyCodeTypeLowerMd4</code>	Safety code type 2: Lower half of MD4.
<code>srcty_kSafetyCodeTypeFullMd4</code>	Safety code type 3: Full MD4.
<code>srcty_kSafetyCodeTypeMax</code>	Max value for safety code enum.

4.14.3 Variable Documentation

4.14.3.1 `srcty_kMinFreeEntriesReceivedBufferForReceive` `const uint32_t srcty_kMinFreeEntriesReceivedBufferForReceive [extern]`

Minimum amount of free entries in the received buffer in case of receiving a message.

One free entry in the received buffer is needed to receive a new message.

4.14.3.2 `srcty_kMinFreeEntriesSendBufferForRetr` `const uint32_t srcty_kMinFreeEntriesSendBufferForRetr [extern]`

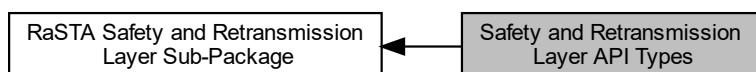
Minimum amount of free entries in the send buffer in case of a retransmission.

In worst case a retransmission of retransmission adds a RetrResp, HB & RetrReq to the send buffer, so there need to be at least 3 free entries to start a retransmission.

4.15 Safety and Retransmission Layer API Types

Type definitions of RaSTA SafRetL API.

Collaboration diagram for Safety and Retransmission Layer API Types:



Classes

- struct `sraty_BufferUtilisation`
Struct for the buffer utilisation of the SafRetL buffers.
- struct `sraty_ConnectionDiagnosticData`
Struct for the diagnostic data of a RaSTA connection.
- struct `sraty_RedundancyChannelDiagnosticData`
Struct for the diagnostic data from a redundancy channel.

Enumerations

- enum `srtty_ConnectionStates` {

 `srtty_kConnectionMin` = 0 , `srtty_kConnectionNotInitialized` = 0 , `srtty_kConnectionClosed` , `srtty_kConnectionDown`

 ,

 `srtty_kConnectionStart` , `srtty_kConnectionUp` , `srtty_kConnectionRetransRequest` , `srtty_kConnectionRetransRunning`

 ,

 `srtty_kConnectionMax` }

 Enum for the state of a RaSTA connection.
- enum `srtty_DiscReason` {

 `srtty_kDiscReasonMin` = 0U , `srtty_kDiscReasonUserRequest` = 0U , `srtty_kDiscReasonNotInUse` = 1U ,

 `srtty_kDiscReasonUnexpectedMessage` = 2U ,

 `srtty_kDiscReasonSequenceNumberError` = 3U , `srtty_kDiscReasonTimeout` = 4U , `srtty_kDiscReasonServiceNotAllowed`

 = 5U , `srtty_kDiscReasonProtocolVersionError` = 6U ,

 `srtty_kDiscReasonRetransmissionFailed` = 7U , `srtty_kDiscReasonProtocolSequenceError` = 8U ,

 `srtty_kDiscReasonMax` }

 Enum for disconnect reason.

4.15.1 Detailed Description

Type definitions of RaSTA SafRetL API.

Specification of the API type structures of the RaSTA Safety and Retransmission Layer.

This module defines the data structures used for the RaSTA SafRetL API interface.

Implements Requirements	RASW-518 Safety and Retransmission Layer Safety Integrity Level RASW-491 Enum Connection States Structure RASW-489 Enum Disc Reason Structure RASW-461 Struct Buffer Utilisation Structure RASW-470 Struct Connection Diagnostic Data Structure RASW-475 Struct Redundancy Channel Diagnostic Data Structure
--------------------------------	---

4.15.2 Enumeration Type Documentation

4.15.2.1 `srtty_ConnectionStates` enum `srtty_ConnectionStates`

Enum for the state of a RaSTA connection.

Implements Requirements	RASW-491 Enum Connection States Structure
--------------------------------	---

Enumerator

<code>srtty_kConnectionMin</code>	Min value for connection state enum.
<code>srtty_kConnectionNotInitialized</code>	Not initialized.
<code>srtty_kConnectionClosed</code>	Connection state closed.
<code>srtty_kConnectionDown</code>	Connection state down.
<code>srtty_kConnectionStart</code>	Connection state start.
<code>srtty_kConnectionUp</code>	Connection state up.
<code>srtty_kConnectionRetransRequest</code>	Connection state retransmission request.
<code>srtty_kConnectionRetransRunning</code>	Connection state retransmission running.

4.15.2.2 `sraty_DiscReason` enum `sraty_DiscReason`

Enum for disconnect reason.

Implements Requirements RASW-489 Enum Disc Reason Structure

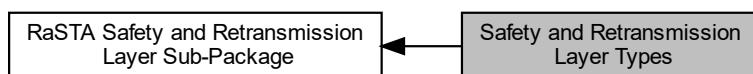
Enumerator

<code>sraty_kDiscReasonMin</code>	Min value for disconnect reason enum.
<code>sraty_kDiscReasonUserRequest</code>	User request.
<code>sraty_kDiscReasonNotInUse</code>	Not in use.
<code>sraty_kDiscReasonUnexpectedMessage</code>	Received message type not expected for the current state.
<code>sraty_kDiscReasonSequenceNumberError</code>	Error in the sequence number verification during connection establishment.
<code>sraty_kDiscReasonTimeout</code>	Timeout for incoming messages.
<code>sraty_kDiscReasonServiceNotAllowed</code>	Service not allowed in this state.
<code>sraty_kDiscReasonProtocolVersionError</code>	Error in the protocol version.
<code>sraty_kDiscReasonRetransmissionFailed</code>	Retransmission failed, requested sequence number not available.
<code>sraty_kDiscReasonProtocolSequenceError</code>	Error in the protocol sequence.
<code>sraty_kDiscReasonMax</code>	Max value for disconnect reason enum.

4.16 Safety and Retransmission Layer Types

Internal type definitions of RaSTA SafRetL.

Collaboration diagram for Safety and Retransmission Layer Types:



Classes

- struct `srtyp_SrMessageHeader`
Typedef for a SafRetL PDU message header.
- struct `srtyp_SrMessageHeaderCreate`
Typedef for SafRetL PDU message header data parameters for creating a message.
- struct `srtyp_SrMessageHeaderUpdate`
Typedef for SafRetL PDU message header data parameters for updating a message header before sending the message.

- struct `srtyp_SrMessage`
Typedef for a SafRetL PDU message.
- struct `srtyp_SrMessagePayload`
Typedef for a SafRetL PDU message payload.
- struct `srtyp_ProtocolVersion`
Typedef for RaSTA protocol version array.

Macros

- `#define SRTYP_PROTOCOL_VERSION_SIZE (4U)`
Size of RaSTA protocol version array.

Enumerations

- enum `srtyp_ConnectionEvents` {
 `srtyp_kConnEventMin` = 0 , `srtyp_kConnEventNone` = 0 , `srtyp_kConnEventOpen` , `srtyp_kConnEventClose` ,
 `srtyp_kConnEventSendData` , `srtyp_kConnEventConnReqReceived` , `srtyp_kConnEventConnRespReceived` ,
 `srtyp_kConnEventRetrReqReceived` ,
 `srtyp_kConnEventRetrRespReceived` , `srtyp_kConnEventDiscReqReceived` , `srtyp_kConnEventHbReceived` ,
 `srtyp_kConnEventDataReceived` ,
 `srtyp_kConnEventRetrDataReceived` , `srtyp_kConnEventSendHb` , `srtyp_kConnEventTimeout` , `srtyp_kConnEventMax`
}

Enum for the events of a RaSTA connection state machine.
- enum `srtyp_SrMessageType` {
 `srtyp_kSrMessageMin` = 6200U , `srtyp_kSrMessageConnReq` = 6200U , `srtyp_kSrMessageConnResp` = 6201U ,
 `srtyp_kSrMessageRetrReq` = 6212U ,
 `srtyp_kSrMessageRetrResp` = 6213U , `srtyp_kSrMessageDiscReq` = 6216U , `srtyp_kSrMessageHb` = 6220U ,
 `srtyp_kSrMessageData` = 6240U ,
 `srtyp_kSrMessageRetrData` = 6241U , `srtyp_kSrMessageMax` = 6300U }

Enum for SafRetL PDU message types.

4.16.1 Detailed Description

Internal type definitions of RaSTA SafRetL.

Specification of the internal type structures of the RaSTA Safety and Retransmission Layer.

This module defines the internal data structures used for the RaSTA SafRetL.

Implements Requirements `RASW-518` Safety and Retransmission Layer Safety Integrity Level
`RASW-560` `sr_state_machine` Events

4.16.2 Enumeration Type Documentation

4.16.2.1 `srtyp_ConnectionEvents` `enum srtyp_ConnectionEvents`

Enum for the events of a RaSTA connection state machine.

Implements Requirements `RASW-560` `sr_state_machine` Events

Enumerator

<code>srtyp_kConnEventMin</code>	Min value for connection events enum.
<code>srtyp_kConnEventNone</code>	No connection event.
<code>srtyp_kConnEventOpen</code>	Open connection event.
<code>srtyp_kConnEventClose</code>	Close connection event.
<code>srtyp_kConnEventSendData</code>	Send data event.
<code>srtyp_kConnEventConnReqReceived</code>	Connection request received event.
<code>srtyp_kConnEventConnRespReceived</code>	Connection response received event.
<code>srtyp_kConnEventRetrReqReceived</code>	Retransmission request received event.
<code>srtyp_kConnEventRetrRespReceived</code>	Retransmission response received event.
<code>srtyp_kConnEventDiscReqReceived</code>	Disconnection request received event.
<code>srtyp_kConnEventHbReceived</code>	Heartbeat received event.
<code>srtyp_kConnEventDataReceived</code>	Data received event.
<code>srtyp_kConnEventRetrDataReceived</code>	Retransmitted data received event.
<code>srtyp_kConnEventSendHb</code>	Send heartbeat event.
<code>srtyp_kConnEventTimeout</code>	Connection timeout event.
<code>srtyp_kConnEventMax</code>	Max value for connection events enum.

4.16.2.2 `srtyp_SrMessageType` enum `srtyp_SrMessageType`

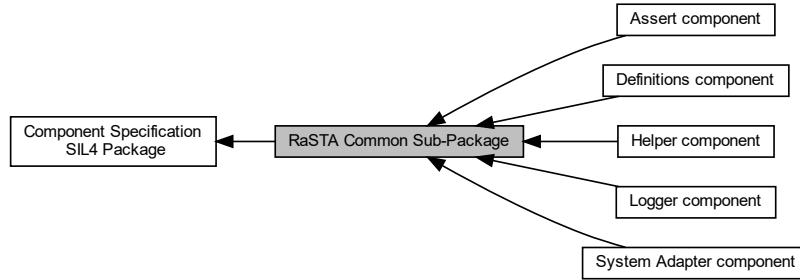
Enum for SafRetL PDU message types.

Enumerator

<code>srtyp_kSrMessageMin</code>	Min value for sr message type enum.
<code>srtyp_kSrMessageConnReq</code>	Connection request message type.
<code>srtyp_kSrMessageConnResp</code>	Connection response message type.
<code>srtyp_kSrMessageRetrReq</code>	Retransmission request message type.
<code>srtyp_kSrMessageRetrResp</code>	Retransmission response message type.
<code>srtyp_kSrMessageDiscReq</code>	Disconnection request message type.
<code>srtyp_kSrMessageHb</code>	Heartbeat message type.
<code>srtyp_kSrMessageData</code>	Data message type.
<code>srtyp_kSrMessageRetrData</code>	Retransmitted data message type.
<code>srtyp_kSrMessageMax</code>	Max value for sr message type enum.

4.17 RaSTA Common Sub-Package

Collaboration diagram for RaSTA Common Sub-Package:



Modules

- [Definitions component](#)
Common definitions for the RaSTA stack implementation.
- [Assert component](#)
Interface of the RaSTA assert functions.
- [Helper component](#)
Interface of the RaSTA helper functions.
- [Logger component](#)
Interface of the RaSTA debug logger.
- [System Adapter component](#)
Interface of the RaSTA system adapter functions.

4.17.1 Detailed Description

Specification of all components within the RaSTA Common Package.

4.18 Definitions component

Common definitions for the RaSTA stack implementation.

Collaboration diagram for Definitions component:



Classes

- struct `radef_TransportChannelDiagnosticData`
Struct for the diagnostic data from a transport channel.

Macros

- `#define PRIVATE static`
Macro for local variables which shall be accessible during unit tests.
- `#define RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS (2U)`
Maximum number of RaSTA connections per RaSTA network.
- `#define RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE (1055U)`
Maximum payload size of a SafRetL PDU message [Bytes].
- `#define RADEF_SR_LAYER_MESSAGE_HEADER_SIZE (28U)`
Header size of a SafRetL PDU message [Bytes].
- `#define RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE (2U)`
Application message length size of a SafRetL PDU message [Bytes].
- `#define RADEF_MAX_SR_LAYER_SAFETY_CODE_SIZE (16U)`
Maximum safety code size of a SafRetL PDU message [Bytes].
- `#define RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE`
Maximum size of a SafRetL PDU message (including header and safety code) [Bytes].
- `#define RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_INTERVALS (5U)`
Number of received message timing distribution diagnostic intervals.
- `#define RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_ARRAY_SIZE (RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_INTERVALS - 1U)`
Size of timing distribution diagnostic interval array. Contains one element less than intervals since last element is set to t_max.
- `#define RADEF_MAX_N_SEND_MAX (20U)`
Maximum number of entries in the received buffer [messages].
- `#define RADEF_SEND_BUFFER_SIZE (RADEF_MAX_N_SEND_MAX)`
Defines the number of send buffer entries [messages].
- `#define RADEF_MAX_NUMBER_OF_RED_CHANNELS (RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS)`
Maximum number of redundancy channels.
- `#define RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS (2U)`
Maximum number of transport channels per redundancy channel.
- `#define RADEF_RED_LAYER_MESSAGE_HEADER_SIZE (8U)`
Header size of a RedL PDU message [Bytes].
- `#define RADEF_MAX_RED_LAYER_CHECK_CODE_SIZE (4U)`
Maximum check code size of a RedL PDU message [Bytes].
- `#define RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE (RADEF_RED_LAYER_MESSAGE_HEADER_SIZE + RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE + RADEF_MAX_RED_LAYER_CHECK_CODE_SIZE)`
Maximum size of RedL PDU message (including RedL header, max. SafRetL PDU message size and max. check code size) [Bytes].
- `#define RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE (RADEF_RED_LAYER_MESSAGE_HEADER_SIZE + RADEF_SR_LAYER_MESSAGE_HEADER_SIZE)`
Minimum size of RedL PDU message (including RedL header, min. SafRetL PDU message size (only SafRetL header) and min. check code size (none)) [Bytes].
- `#define RADEF_MAX_DEFER_QUEUE_SIZE (10U)`
Maximum size of a redundancy channel defer queue [messages].
- `#define RADEF_MAX_RED_LAYER_N_DIAGNOSIS (1000U)`
Maximum RedL diagnosis window size (Ndiagnosis) [messages].

Enumerations

- enum `radef_RaStaReturnCode` {
 `radef_kMin` = 0 , `radef_kNoError` = 0 , `radef_kNoMessageReceived` = 1 , `radef_kNoMessageToSend` = 2 ,
 `radef_kNotInitialized` = 3 , `radef_kAlreadyInitialized` = 4 , `radef_kInvalidConfiguration` = 5 , `radef_kInvalidParameter` = 6 ,
 `radef_kInvalidMessageType` = 7 , `radef_kInvalidMessageSize` = 8 , `radef_kInvalidBufferSize` = 9 ,
 `radef_kInvalidMessageCrc` = 10 ,
 `radef_kInvalidMessageMd4` = 11 , `radef_kReceiveBufferFull` = 12 , `radef_kDeferQueueEmpty` = 13 ,
 `radef_kSendBufferFull` = 14 ,
 `radef_kInvalidSequenceNumber` = 15 , `radef_kInternalError` = 16 , `radef_kInvalidOperationInCurrentState` = 17 , `radef_kMax` }

Enum for function return codes of the RaSTA stack.

4.18.1 Detailed Description

Common definitions for the RaSTA stack implementation.

Specification of the common definitions of the RaSTA Stack.

This module defines the common definitions, types and data structures used by both RaSTA layers (SafRetL & RedL).

Implements Requirements [RASW-525](#) Component rasta_definitions Overview
[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

4.18.2 Macro Definition Documentation

4.18.2.1 RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE `#define RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE`

Value:

```
(RADEF_SR_LAYER_MESSAGE_HEADER_SIZE + RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE +
RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE + \
RADEF_MAX_SR_LAYER_SAFETY_CODE_SIZE)
```

Maximum size of a SafRetL PDU message (including header and safety code) [Bytes].

4.18.2.2 RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE `#define RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE (2U)`

Application message length size of a SafRetL PDU message [Bytes].

Embedded length of an application message inside a SafRetL PDU message, as stated in chapter 5.5.10 of "Check code" of the standard "Elektrische Bahn-Signalanlagen – Teil 200: Sicheres Übertragungsprotokoll RaSTA nach DIN EN 50159 (VDE0831-159); Juni 2015".

4.18.3 Enumeration Type Documentation

4.18.3.1 `radef_RaStaReturnCode` enum `radef_RaStaReturnCode`

Enum for function return codes of the RaSTA stack.

Implements Requirements [RASW-483](#) Enum RaSta Return Code Structure
[RASW-503](#) Enum RaSta Return Code Usage

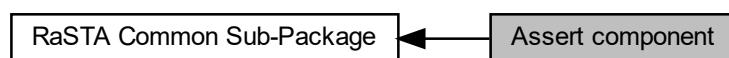
Enumerator

radef_kMin	Min value for RaSTA return code enum.
radef_kNoError	No error.
radef_kNoMessageReceived	No message received.
radef_kNoMessageToSend	No message to send.
radef_kNotInitialized	Not initialized.
radef_kAlreadyInitialized	Already initialized.
radef_kInvalidConfiguration	Invalid configuration.
radef_kInvalidParameter	Invalid parameter.
radef_kInvalidMessageType	Invalid message type.
radef_kInvalidMessageSize	Invalid message size.
radef_kInvalidBufferSize	Invalid buffer size.
radef_kInvalidMessageCrc	Invalid message crc.
radef_kInvalidMessageMd4	Invalid message MD4.
radef_kReceiveBufferFull	Receive buffer full.
radef_kDeferQueueEmpty	Defer queue empty.
radef_kSendBufferFull	Send buffer full.
radef_kInvalidSequenceNumber	Invalid sequence number.
radef_kInternalError	Internal error.
radef_kInvalidOperationInCurrentState	Invalid operation in the current state.
radef_kMax	Max value for RaSTA return code enum.

4.19 Assert component

Interface of the RaSTA assert functions.

Collaboration diagram for Assert component:

**Functions**

- void `raas_AssertNotNull` (const void *const pointer, const `radef_RaStaReturnCode` error_reason)
Assert if a pointer is not NULL.
- void `raas_AssertTrue` (const bool condition, const `radef_RaStaReturnCode` error_reason)
Assert a bool condition is true.
- void `raas_AssertU8InRange` (const uint8_t value, const uint8_t min_value, const uint8_t max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a uint8_t value is in a defined range.

- void `raas_AssertU16InRange` (const uint16_t value, const uint16_t min_value, const uint16_t max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a uint16_t value is in a defined range.
- void `raas_AssertU32InRange` (const uint32_t value, const uint32_t min_value, const uint32_t max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a uint32_t value is in a defined range.

4.19.1 Detailed Description

Interface of the RaSTA assert functions.

Specification of the Assert component of the RaSTA Stack.

This module provides all necessary assert functions to perform checks and throw a specific fatal error if the check fails.

Implements Requirements [RASW-533](#) Component rasta_assert Overview
[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

4.19.2 Function Documentation

4.19.2.1 `raas_AssertNotNull()` void `raas_AssertNotNull` (
 const void *const *pointer*,
 const `radef_RaStaReturnCode` *error_reason*)

Assert if a pointer is not NULL.

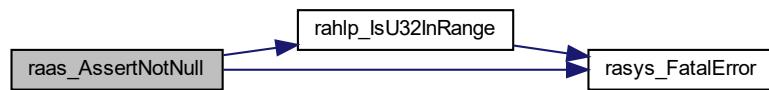
This function checks if a pointer is not NULL. If that is not the case, the passed error code is thrown as fatal error using `rasyFatalError`.

Implements Requirements [RASW-534](#) Assert not Null Function
[RASW-521](#) Input Parameter Check

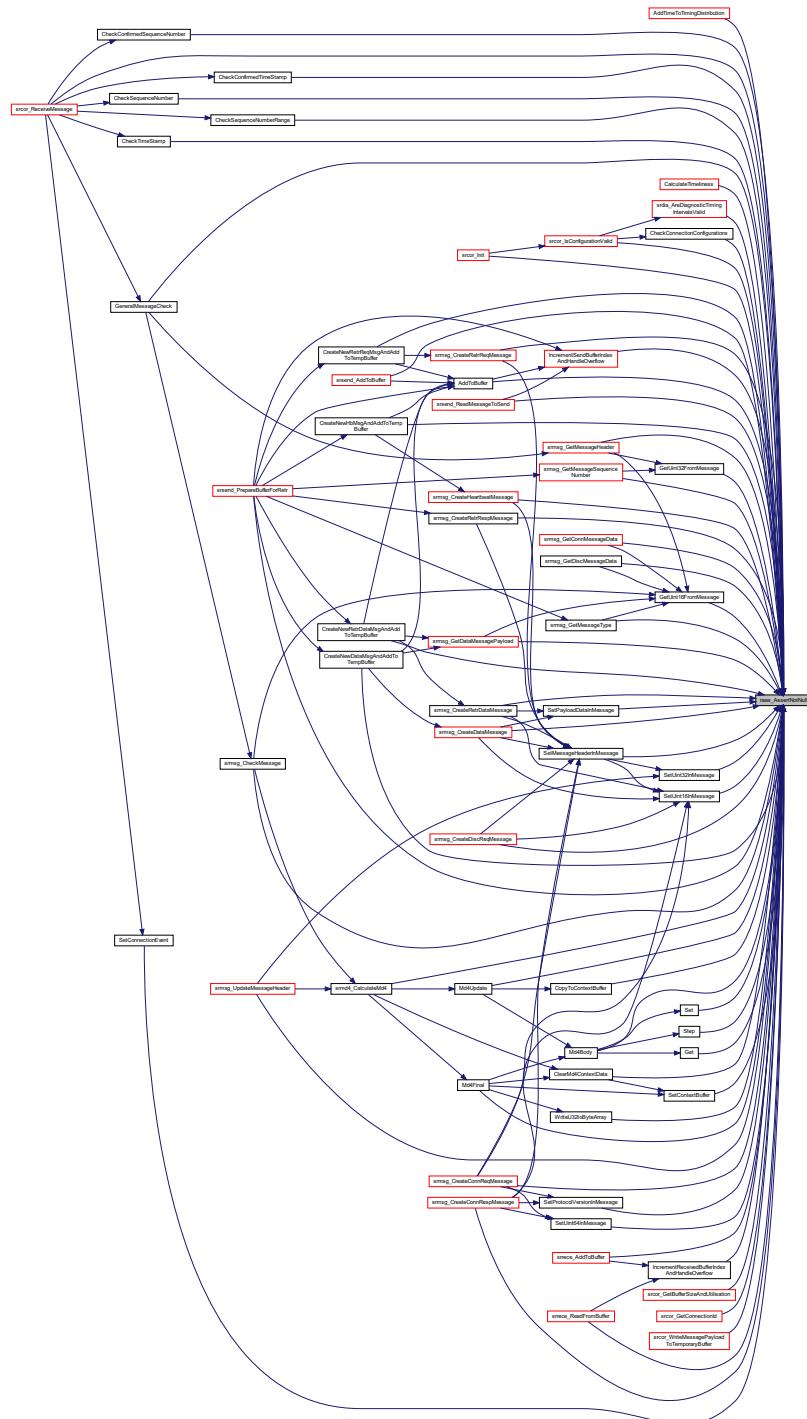
Parameters

in	<i>pointer</i>	Pointer to check.
in	<i>error_reason</i>	Reason in case of error. Valid range: <code>radef_kMin</code> <= value < <code>radef_kMax</code> .

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.19.2.2 raas_ASSERT() void raas_ASSERT ( const bool condition, const raedef RaStaReturnCode error reason )
```

Assert a bool condition is true.

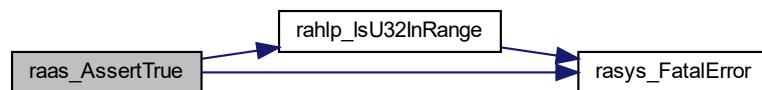
This function checks if a bool condition is true. If that is not the case, the passed error code is thrown as fatal error using [rasys_FatalError](#).

Implements Requirements [RASW-535](#) Assert True Function
[RASW-521](#) Input Parameter Check

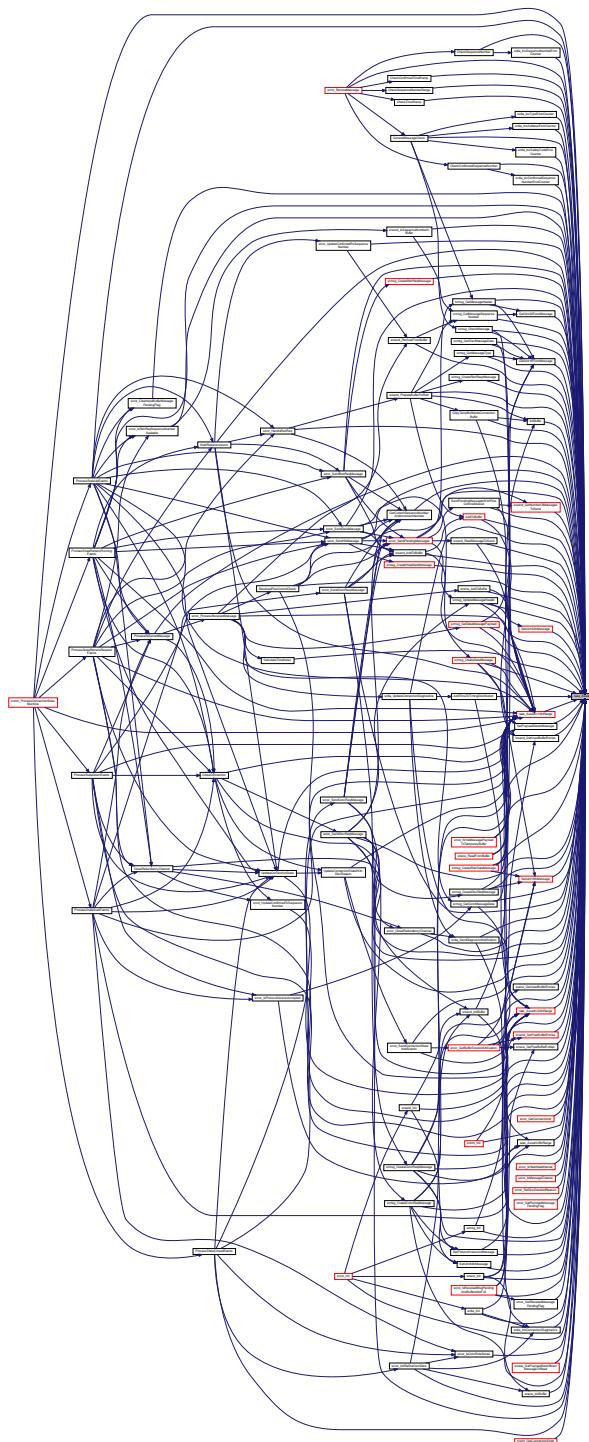
Parameters

in	<i>condition</i>	Condition to check.
in	<i>error_reason</i>	Reason in case of error. Valid range: <code>raodef_kMin</code> <= value < <code>raodef_kMax</code> .

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.19.2.3 raas_AssertU16InRange() void raas_AssertU16InRange (
    const uint16_t value,
    const uint16_t min_value,
    const uint16_t max_value,
    const radef_RaStaReturnCode error_reason )
```

Assert if a uint16_t value is in a defined range.

This function checks if a uint16_t value is inside a defined range. Check condition: `min_value <= value <= max_value`. The minimum value must be smaller or equal to the maximum value, otherwise a [radef_kInvalidParameter](#) fatal error is thrown.

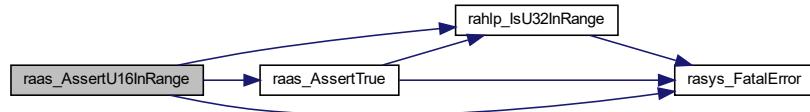
Implements Requirements [RASW-536](#) Assert U16 in Range Function

[RASW-521](#) Input Parameter Check

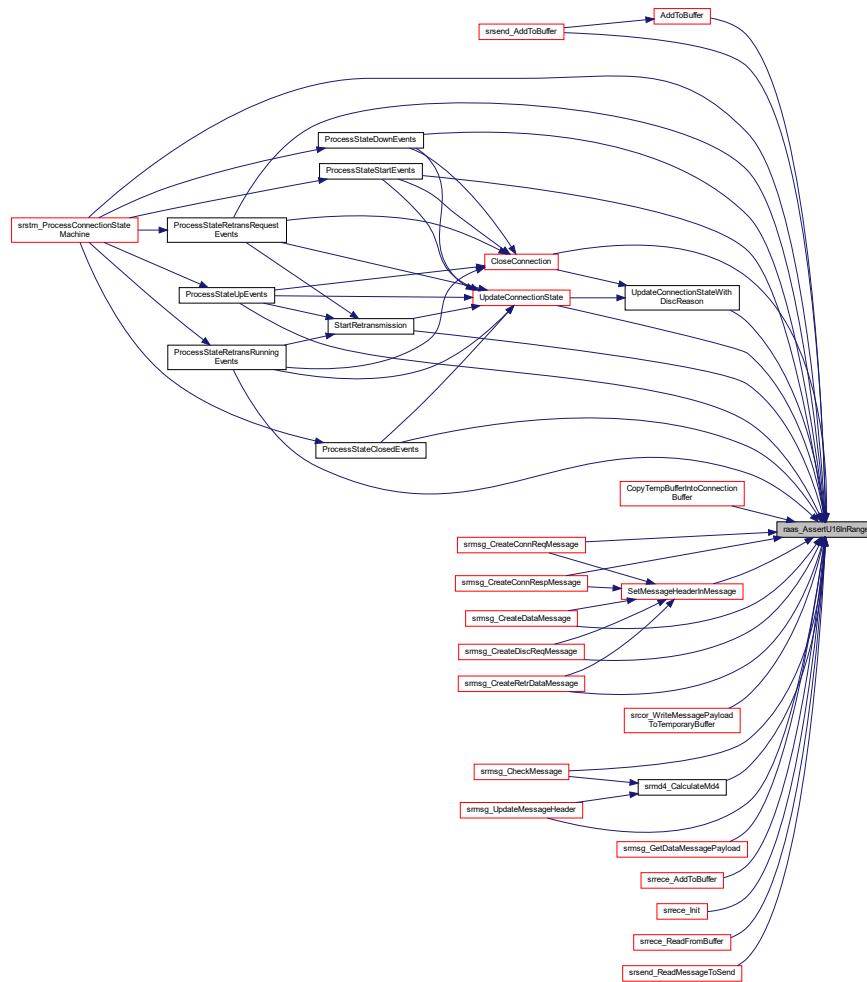
Parameters

in	<i>value</i>	Value to check. The full value range is valid and usable.
in	<i>min_value</i>	Minimum value for the check. The full value range is valid and usable.
in	<i>max_value</i>	Maximum value for the check. The full value range is valid and usable.
in	<i>error_reason</i>	Reason in case of error. Valid range: radef_kMin <= value < radef_kMax .

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.19.2.4 raas_AssertU32InRange() void raas_AssertU32InRange (
    const uint32_t value,
    const uint32_t min_value,
    const uint32_t max_value,
    const radef\_RaStaReturnCode error_reason )
```

Assert if a `uint32_t` value is in a defined range.

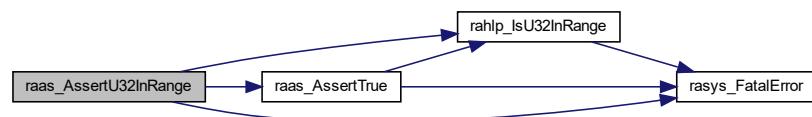
This function checks if a `uint32_t` value is inside a defined range. Check condition: `min_value <= value <= max_value`. The minimum value must be smaller or equal to the maximum value, otherwise a [radef_klInvalidParameter](#) fatal error is thrown.

Implements Requirements [RASW-537](#) Assert U32 in Range Function
[RASW-521](#) Input Parameter Check

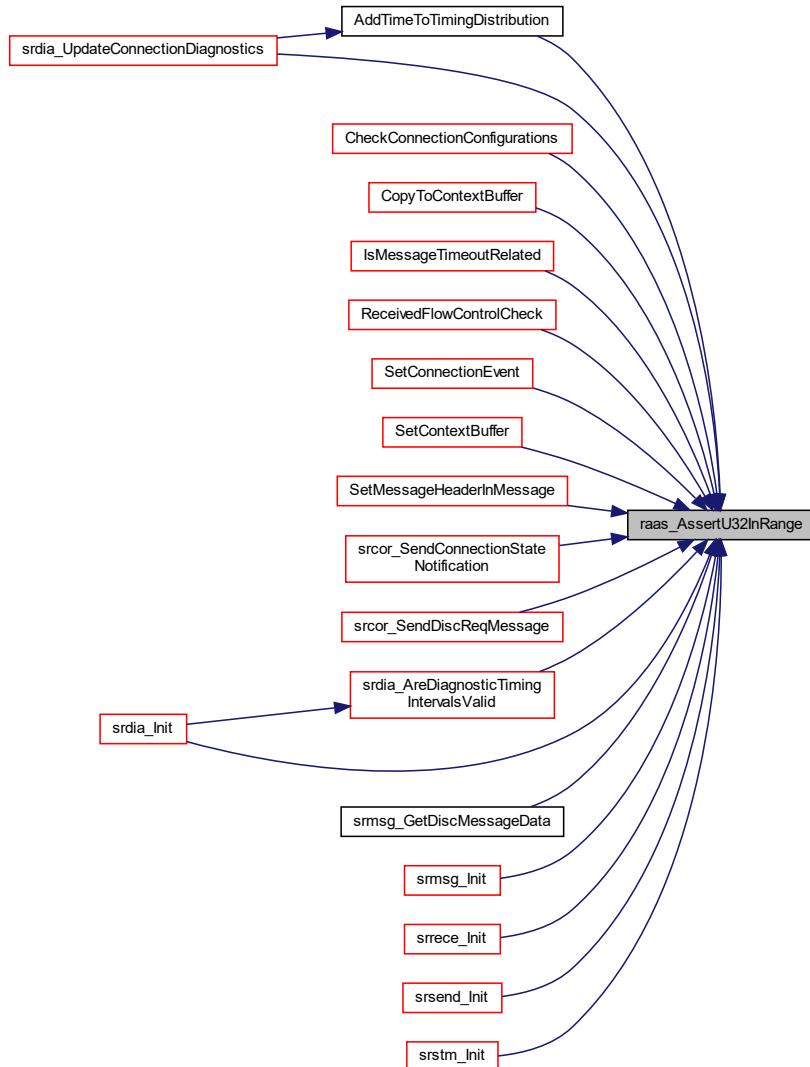
Parameters

in	<i>value</i>	Value to check. The full value range is valid and usable.
in	<i>min_value</i>	Minimum value for the check. The full value range is valid and usable.
in	<i>max_value</i>	Maximum value for the check. The full value range is valid and usable.
in	<i>error_reason</i>	Reason in case of error. Valid range: <code>radef_kMin</code> <= value < <code>radef_kMax</code> .

Here is the call graph for this function:



Here is the caller graph for this function:



4.19.2.5 raas_AssertU8InRange() `void raas_AssertU8InRange (`
 `const uint8_t value,`
 `const uint8_t min_value,`
 `const uint8_t max_value,`
 `const raedef_RaStaReturnCode error_reason)`

Assert if a `uint8_t` value is in a defined range.

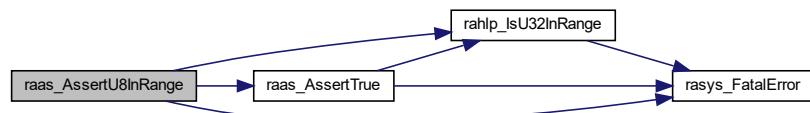
This function checks if a `uint8_t` value is inside a defined range. Check condition: `min_value <= value <= max_value`. The minimum value must be smaller or equal to the maximum value, otherwise a `raedef_kInvalidParameter` fatal error is thrown.

Implements Requirements [RASW-538](#) Assert U8 in Range Function
[RASW-521](#) Input Parameter Check

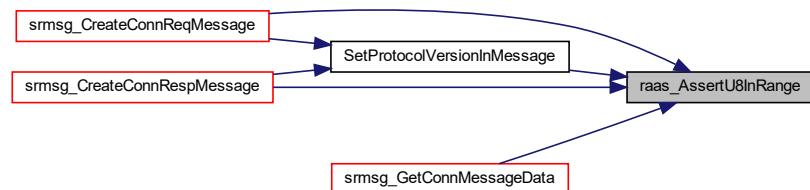
Parameters

in	<i>value</i>	Value to check. The full value range is valid and usable.
in	<i>min_value</i>	Minimum value for the check. The full value range is valid and usable.
in	<i>max_value</i>	Maximum value for the check. The full value range is valid and usable.
in	<i>error_reason</i>	Reason in case of error. Valid range: <code>radef_kMin</code> <= <i>value</i> < <code>radef_kMax</code> .

Here is the call graph for this function:



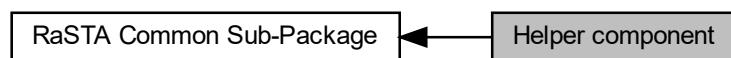
Here is the caller graph for this function:



4.20 Helper component

Interface of the RaSTA helper functions.

Collaboration diagram for Helper component:



Functions

- bool `rahlp_IsU16InRange` (const uint16_t *value*, const uint16_t *min_value*, const uint16_t *max_value*)
Checks if a uint16_t value is in a defined range.
- bool `rahlp_IsU32InRange` (const uint32_t *value*, const uint32_t *min_value*, const uint32_t *max_value*)
Checks if a uint32_t value is in a defined range.

4.20.1 Detailed Description

Interface of the RaSTA helper functions.

Specification of the Helper component of the RaSTA Stack.

This module provides some helper functions for common used functionalities.

Implements Requirements [RASW-818](#) Component rasta_Helper Overview
[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

4.20.2 Function Documentation

4.20.2.1 rahlp_IsU16InRange() `bool rahlp_IsU16InRange (`
 `const uint16_t value,`
 `const uint16_t min_value,`
 `const uint16_t max_value)`

Checks if a `uint16_t` value is in a defined range.

This helper function checks if a `uint16_t` value is inside a defined range. Check condition: `min_value <= value <= max_value`. The minimum value must be smaller or equal to the maximum value, otherwise a [radef_kInvalidParameter](#) fatal error is thrown.

Implements Requirements [RASW-821](#) Is U16 in Range Function
[RASW-521](#) Input Parameter Check

Parameters

<code>in</code>	<code>value</code>	Value to check. The full value range is valid and usable.
<code>in</code>	<code>min_value</code>	Minimum value for the check. The full value range is valid and usable.
<code>in</code>	<code>max_value</code>	Maximum value for the check. The full value range is valid and usable.

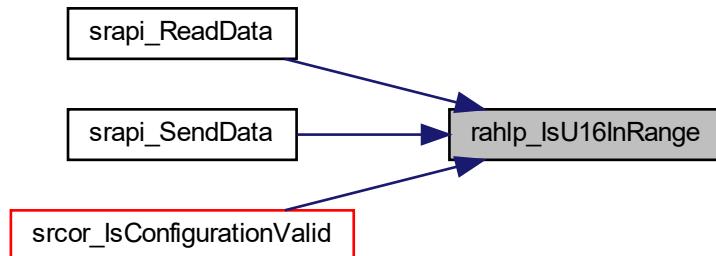
Returns

true -> successful operation, value inside the range
false -> value outside the range

Here is the call graph for this function:



Here is the caller graph for this function:



```
4.20.2.2 rahlp_IsU32InRange() bool rahlp_IsU32InRange ( const uint32_t value, const uint32_t min_value, const uint32_t max_value )
```

Checks if a uint32_t value is in a defined range.

This helper function checks if a uint32_t value is inside a defined range. Check condition: `min_value <= value <= max_value`. The minimum value must be smaller or equal to the maximum value, otherwise a [radef_kInvalidParameter](#) fatal error is thrown.

Implements Requirements [RASW-820](#) Is U32 in Range Function
[RASW-521](#) Input Parameter Check

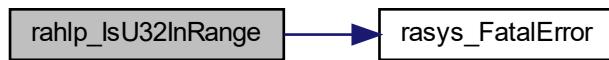
Parameters

in	<i>value</i>	Value to check. The full value range is valid and usable.
in	<i>min_value</i>	Minimum value for the check. The full value range is valid and usable.
in	<i>max_value</i>	Maximum value for the check. The full value range is valid and usable.

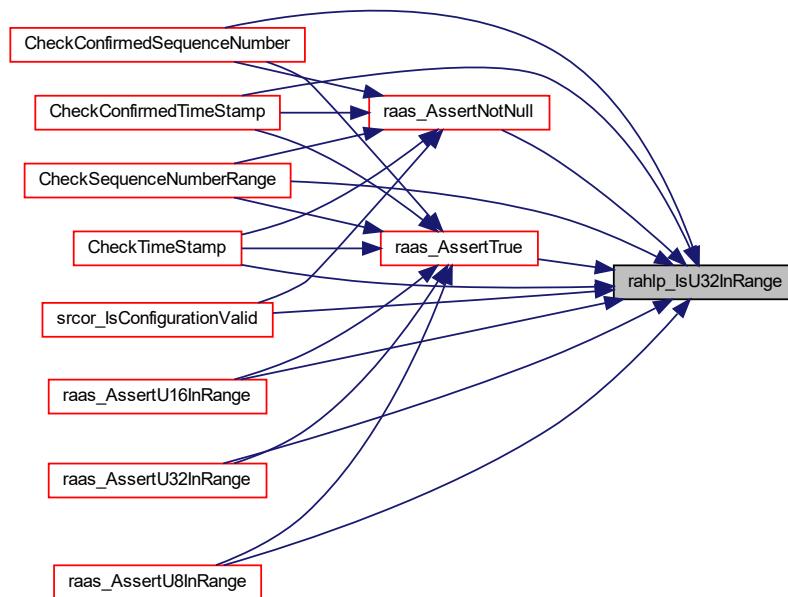
Returns

true -> successful operation, value inside the range
 false -> value outside the range

Here is the call graph for this function:



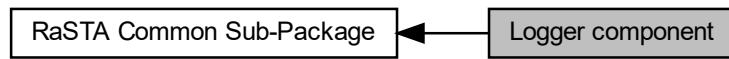
Here is the caller graph for this function:



4.21 Logger component

Interface of the RaSTA debug logger.

Collaboration diagram for Logger component:



Macros

- `#define ralog_ENABLE_LOGGER 0`
Global logger disable.
- `#define ralog_INIT_LOGGER(log_level) 0U`
Empty placeholder macro for `ralog_INIT_LOGGER()` used for RELEASE build.
- `#define ralog_LOG_ERROR(logger_id, message, ...)`
Empty placeholder macro for `ralog_LOG_ERROR()` used for RELEASE build.
- `#define ralog_LOG_WARN(logger_id, message, ...)`
Empty placeholder macro for `ralog_LOG_WARN()` used for RELEASE build.
- `#define ralog_LOG_INFO(logger_id, message, ...)`
Empty placeholder macro for `ralog_LOG_INFO()` used for RELEASE build.
- `#define ralog_LOG_DEBUG(logger_id, message, ...)`
Empty placeholder macro for `ralog_LOG_DEBUG()` used for RELEASE build.

4.21.1 Detailed Description

Interface of the RaSTA debug logger.

Specification of the Logger component of the RaSTA Stack.

This module provides different logging utilities to log debug information on different log levels. The logger output is written to stdout by printf. The logger module can be enabled for the DEBUG build by defining: `ralog_ENABLE_LOGGER 1`. The logger module must be disabled for the RELEASE build by defining: `ralog_ENABLE_LOGGER 0`. This removes the logger code completely from the RELEASE build by the use of macros.

Implements Requirements [RASW-540](#) Component rasta_logger Overview

[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

4.21.2 Macro Definition Documentation

4.21.2.1 ralog_ENABLE_LOGGER #define ralog_ENABLE_LOGGER 0

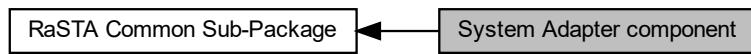
Global logger disable.

ralog_ENABLE_LOGGER 0 -> logger disabled, used for RELEASE build, no logger code is generated

4.22 System Adapter component

Interface of the RaSTA system adapter functions.

Collaboration diagram for System Adapter component:



Functions

- `uint32_t rasys_GetTimerValue (void)`
Returns the actual value of a free running up counting timer.
- `uint32_t rasys_GetTimerGranularity (void)`
Returns the granularity of the free running up counting timer.
- `uint32_t rasys_GetRandomNumber (void)`
Returns a random generated number within the uint32_t type range.
- `void rasys_FatalError (const radef_RaStaReturnCode error_reason)`
Fatal error function.

4.22.1 Detailed Description

Interface of the RaSTA system adapter functions.

Specification of the System Adapter component of the RaSTA Stack.

This module defines the interface to the necessary system functions used by the SW. This includes functionalities related to time, fatal error handling and random number generation. The RaSTA common only defines the interface, the implementation of this system adapter interface functions must be done by the system integrator.

Remarks

The error handling for all function must be implemented and handled by the system integrator when developing the SafRetL adapter.

Implements Requirements [RASW-527](#) Component rasta_system_adapter Overview

[RASW-518](#) Safety and Retransmission Layer Safety Integrity Level

4.22.2 Function Documentation

4.22.2.1 rasys_FatalError() `void rasys_FatalError (`
 `const radef_RaStaReturnCode error_reason)`

Fatal error function.

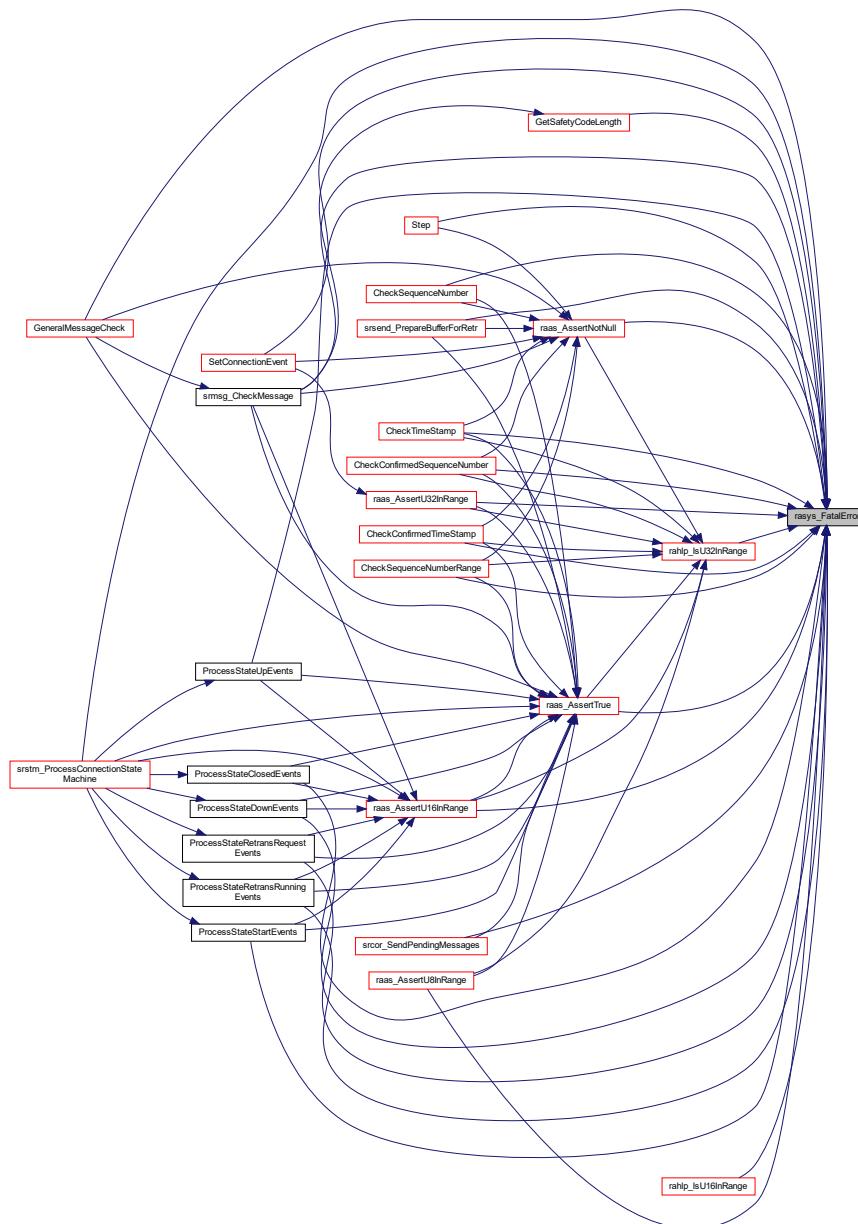
This function returns the program execution to the operating system. This function is called in case of a fatal internal error. Important: This function is not allowed to return.

Implements Requirements [RASW-528](#) Fatal Error Function
[RASW-417](#) Fatal Error Handling Function Structure
[RASW-416](#) Error Code
[RASW-503](#) Enum RaSta Return Code Usage
[RASW-520](#) Error Handling

Parameters

in	<i>error_reason</i>	Reason of the fatal error. Valid range: <code>radef_kMin <= value < radef_kMax</code> .
----	---------------------	---

Here is the caller graph for this function:



4.22.2.2 **rasy_GetRandomNumber()**

```
uint32_t rasy_GetRandomNumber (
    void )
```

Returns a random generated number within the `uint32_t` type range.

The value is used to randomize sequence number at startup. There is no cryptographic function which relies on that value. Therefore, a simple algorithm with a different seed value at startup is sufficient.

Implements Requirements [RASW-529](#) Get Random Number Function
[RASW-414](#) Get Random Number Function Structure
[RASW-413](#) Random Number

Returns

uint32_t Random number

Here is the caller graph for this function:



4.22.2.3 rasys_GetTimerGranularity() uint32_t rasys_GetTimerGranularity (void)

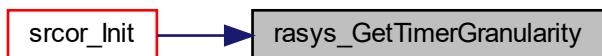
Returns the granularity of the free running up counting timer.

Implements Requirements [RASW-530](#) Get Timer Granularity Function
[RASW-420](#) Get Timer Granularity Function Structure
[RASW-419](#) Timer Granularity

Returns

uint32_t Granularity of the timer [ms].

Here is the caller graph for this function:



4.22.2.4 rasys_GetTimerValue() `uint32_t rasys_GetTimerValue (`
`void)`

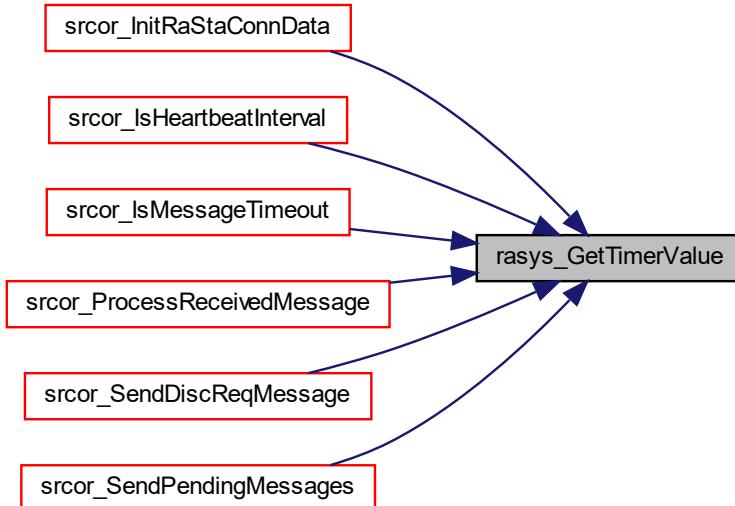
Returns the actual value of a free running up counting timer.

Implements Requirements [RASW-531](#) Get Timer Value Function
[RASW-410](#) Get Timer Value Function Structure
[RASW-422](#) Timer Value

Returns

`uint32_t` Time [ms]. The full range of the `uint32_t` type is used.

Here is the caller graph for this function:



5 Class Documentation

5.1 Md4Context Struct Reference

Structure holding the context data for a MD4 calculation.

Public Attributes

- `uint32_t byte_count_low`
Data size byte count lower 29 bits [bytes], will be multiplied by 8 to get the number of bits.
- `uint32_t bit_count_high`
Data size bit count upper 32 bits [bits].
- `uint32_t a`
A part of the calculated hash.
- `uint32_t b`
B part of the calculated hash.
- `uint32_t c`
C part of the calculated hash.
- `uint32_t d`
D part of the calculated hash.
- `uint8_t buffer [MD4_INPUT_DATA_BLOCK_SIZE]`
Internal buffer for remaining data size < 64 bytes.
- `uint32_t block [MD4_INPUT_DATA_BLOCK_SIZE/BYTES_PER_U32]`
Internal 32 bit aligned data block.

5.1.1 Detailed Description

Structure holding the context data for a MD4 calculation.

The documentation for this struct was generated from the following file:

- [srmd4_sr_md4.c](#)

5.2 `radef_TransportChannelDiagnosticData` Struct Reference

Struct for the diagnostic data from a transport channel.

```
#include <raodef_rasta_definitions.h>
```

Public Attributes

- `uint32_t n_diagnosis`
Diagnosis window size [messages]. Valid range: 0 <= value <= Configured value of n_diagnosis in RedL configuration.
- `uint32_t n_missed`
Nmissed [messages]. Number of messages which are not received on this transport channel within Tseq from the first reception on an other transport channel. Valid range: 0 <= value <= configured value of n_diagnosis in RedL configuration.
- `uint32_t t_drift`
Tdrift [ms]. Sum of the delays of received messages in relation to the fastest transport channel. Full value range is valid and usable.
- `uint32_t t_drift2`
Tdrift2 [ms^2]. Sum of the squares of the delays of received messages in relation to the fastest transport channel. Full value range is valid and usable.

5.2.1 Detailed Description

Struct for the diagnostic data from a transport channel.

This structure is defined in the common part because it is used by both layers. The RedL passes its diagnostic data to the SafRetL using this structure.

Implements Requirements [RASW-474](#) Struct Transport Channel Diagnostic Data Structure

5.2.2 Member Data Documentation

5.2.2.1 n_diagnosis `uint32_t radef_TransportChannelDiagnosticData::n_diagnosis`

Diagnosis window size [messages]. Valid range: 0 <= value <= Configured value of n_diagnosis in RedL configuration.

Implements Requirements [RASW-469](#) N diagnosis

5.2.2.2 n_missed `uint32_t radef_TransportChannelDiagnosticData::n_missed`

Nmissed [messages]. Number of messages which are not received on this transport channel within Tseq from the first reception on an other transport channel. Valid range: 0 <= value <= configured value of n_diagnosis in RedL configuration.

Implements Requirements [RASW-473](#) N missed

5.2.2.3 t_drift `uint32_t radef_TransportChannelDiagnosticData::t_drift`

Tdrift [ms]. Sum of the delays of received messages in relation to the fastest transport channel. Full value range is valid and usable.

Implements Requirements [RASW-472](#) T drift

5.2.2.4 t_drift2 uint32_t radef_TransportChannelDiagnosticData::t_drift2

Tdrift2 [ms²]. Sum of the squares of the delays of received messages in relation to the fastest transport channel. Full value range is valid and usable.

Implements Requirements RASW-467 T drift2

The documentation for this struct was generated from the following file:

- [radef_rasta_definitions.h](#)

5.3 sraty_BufferUtilisation Struct Reference

Struct for the buffer utilisation of the SafRetL buffers.

```
#include <sraty_sr_api_types.h>
```

Public Attributes

- `uint16_t send_buffer_used`
Used send buffer entries [messages]. Valid range: 0 <= value <= RADEF_SEND_BUFFER_SIZE.
- `uint16_t send_buffer_free`
Free send buffer entries [messages]. Valid range: 0 <= value <= RADEF_SEND_BUFFER_SIZE.
- `uint16_t receive_buffer_used`
Used receive buffer entries [messages]. Valid range: 0 <= value <= configured receive buffer size (NsendMax).
- `uint16_t receive_buffer_free`
Free receive buffer entries [messages]. Valid range: 0 <= value <= configured receive buffer size (NsendMax).

5.3.1 Detailed Description

Struct for the buffer utilisation of the SafRetL buffers.

Implements Requirements RASW-461 Struct Buffer Utilisation Structure

5.3.2 Member Data Documentation

5.3.2.1 receive_buffer_free uint16_t sraty_BufferUtilisation::receive_buffer_free

Free receive buffer entries [messages]. Valid range: 0 <= value <= configured receive buffer size (NsendMax).

Implements Requirements RASW-463 Free Receive Buffer Entries

5.3.2.2 receive_buffer_used `uint16_t sraty_BufferUtilisation::receive_buffer_used`

Used receive buffer entries [messages]. Valid range: $0 \leq \text{value} \leq \text{Configured receive buffer size (NsendMax)}$.

Implements Requirements `RASW-4 64` Used Receive Buffer Entries

5.3.2.3 send_buffer_free `uint16_t sraty_BufferUtilisation::send_buffer_free`

Free send buffer entries [messages]. Valid range: $0 \leq \text{value} \leq \text{RADEF_SEND_BUFFER_SIZE}$.

Implements Requirements `RASW-4 65` Free Send Buffer Entries

5.3.2.4 send_buffer_used `uint16_t sraty_BufferUtilisation::send_buffer_used`

Used send buffer entries [messages]. Valid range: $0 \leq \text{value} \leq \text{RADEF_SEND_BUFFER_SIZE}$.

Implements Requirements `RASW-4 60` Used Send Buffer Entries

The documentation for this struct was generated from the following file:

- `sraty_sr_api_types.h`

5.4 sraty_ConnectionDiagnosticData Struct Reference

Struct for the diagnostic data of a RaSTA connection.

```
#include <sraty_sr_api_types.h>
```

Public Attributes

- `uint32_t ec_safety`
Error counter for safety code check failed [messages]. Full value range is valid and usable.
- `uint32_t ec_address`
Error counter for implausible address [messages]. Full value range is valid and usable.
- `uint32_t ec_type`
Error counter for undefined message type [messages]. Full value range is valid and usable.
- `uint32_t ec_sn`
Error counter for implausible sequence number [messages]. Full value range is valid and usable.
- `uint32_t ec_csn`
Error counter for implausible confirmed sequence number [messages]. Full value range is valid and usable.
- `uint32_t t_rtd_distribution [RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_INTERVALS]`
Distribution of round trip delay time [messages]. Valid range: $0 \leq \text{value} \leq \text{Configured value of n_diag_window in SafRetL configuration}$.
- `uint32_t t_alive_distribution [RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_INTERVALS]`
Distribution of alive time [messages]. Valid range: $0 \leq \text{value} \leq \text{Configured value of n_diag_window in SafRetL configuration}$.

5.4.1 Detailed Description

Struct for the diagnostic data of a RaSTA connection.

Implements Requirements [RASW-470](#) Struct Connection Diagnostic Data Structure

5.4.2 Member Data Documentation

5.4.2.1 `ec_address` `uint32_t sraty_ConnectionDiagnosticData::ec_address`

Error counter for implausible address [messages]. Full value range is valid and usable.

Implements Requirements [RASW-482](#) EC address

5.4.2.2 `ec_csn` `uint32_t sraty_ConnectionDiagnosticData::ec_csn`

Error counter for implausible confirmed sequence number [messages]. Full value range is valid and usable.

Implements Requirements [RASW-479](#) EC CSN

5.4.2.3 `ec_safety` `uint32_t sraty_ConnectionDiagnosticData::ec_safety`

Error counter for safety code check failed [messages]. Full value range is valid and usable.

Implements Requirements [RASW-468](#) EC safety

5.4.2.4 `ec_sn` `uint32_t sraty_ConnectionDiagnosticData::ec_sn`

Error counter for implausible sequence number [messages]. Full value range is valid and usable.

Implements Requirements [RASW-480](#) EC SN

5.4.2.5 ec_type `uint32_t sraty_ConnectionDiagnosticData::ec_type`

Error counter for undefined message type [messages]. Full value range is valid and usable.

Implements Requirements [RASW-481](#) EC type

5.4.2.6 t_alive_distribution `uint32_t sraty_ConnectionDiagnosticData::t_alive_distribution[RADEF_DIAGNOSTIC_TIMING_D]`

Distribution of alive time [messages]. Valid range: $0 \leq \text{value} \leq \text{Configured value of n_diag_window in SafRetL configuration}$.

Implements Requirements [RASW-477](#) Alive Time Distribution

5.4.2.7 t_rtd_distribution `uint32_t sraty_ConnectionDiagnosticData::t_rtd_distribution[RADEF_DIAGNOSTIC_TIMING_D]`

Distribution of round trip delay time [messages]. Valid range: $0 \leq \text{value} \leq \text{Configured value of n_diag_window in SafRetL configuration}$.

Implements Requirements [RASW-478](#) Round Trip Delay Time Distribution

The documentation for this struct was generated from the following file:

- `sraty_sr_api_types.h`

5.5 sraty_RedundancyChannelDiagnosticData Struct Reference

Struct for the diagnostic data from a redundancy channel.

```
#include <sraty_sr_api_types.h>
```

Public Attributes

- `uint32_t transport_channel_id`
*Transport channel identification. Valid range: $0 \leq \text{value} < \text{RADEF_MAX_NUMBER_OF_RED_CHANNELS} * \text{RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS}$.*
- `uint32_t n_diagnosis`
Diagnosis window size [messages]. Valid range: $0 \leq \text{value} \leq \text{Configured value of n_diagnosis in RedL configuration}$.
- `uint32_t n_missed`
Nmissed [messages]. Number of messages which are not received on this transport channel within Tseq from the first reception on an other transport channel. Valid range: $0 \leq \text{value} \leq \text{configured value of n_diagnosis in RedL configuration}$.
- `uint32_t t_drift`
Tdrift [ms]. Sum of the delays of received messages in relation to the fastest transport channel. Full value range is valid and usable.
- `uint32_t t_drift2`
Tdrift2 [ms²]. Sum of the squares of the delays of received messages in relation to the fastest transport channel. Full value range is valid and usable.

5.5.1 Detailed Description

Struct for the diagnostic data from a redundancy channel.

Implements Requirements [RASW-475](#) Struct Redundancy Channel Diagnostic Data Structure

5.5.2 Member Data Documentation

5.5.2.1 `n_diagnosis` `uint32_t sraty_RedundancyChannelDiagnosticData::n_diagnosis`

Diagnosis window size [messages]. Valid range: $0 \leq \text{value} \leq$ Configured value of `n_diagnosis` in RedL configuration.

Implements Requirements [RASW-469](#) N diagnosis

5.5.2.2 `n_missed` `uint32_t sraty_RedundancyChannelDiagnosticData::n_missed`

Nmissed [messages]. Number of messages which are not received on this transport channel within Tseq from the first reception on an other transport channel. Valid range: $0 \leq \text{value} \leq$ configured value of `n_diagnosis` in RedL configuration.

Implements Requirements [RASW-473](#) N missed

5.5.2.3 `t_drift` `uint32_t sraty_RedundancyChannelDiagnosticData::t_drift`

Tdrift [ms]. Sum of the delays of received messages in relation to the fastest transport channel. Full value range is valid and usable.

Implements Requirements [RASW-472](#) T drift

5.5.2.4 `t_drift2` `uint32_t sraty_RedundancyChannelDiagnosticData::t_drift2`

Tdrift2 [ms^2]. Sum of the squares of the delays of received messages in relation to the fastest transport channel. Full value range is valid and usable.

Implements Requirements [RASW-467](#) T drift2

5.5.2.5 `transport_channel_id` `uint32_t sraty_RedundancyChannelDiagnosticData::transport_channel_id`

Transport channel identification. Valid range: $0 \leq \text{value} < \text{RADEF_MAX_NUMBER_OF_RED_CHANNELS} * \text{RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS}$.

Implements Requirements [RASW-471](#) Transport Channel Id

The documentation for this struct was generated from the following file:

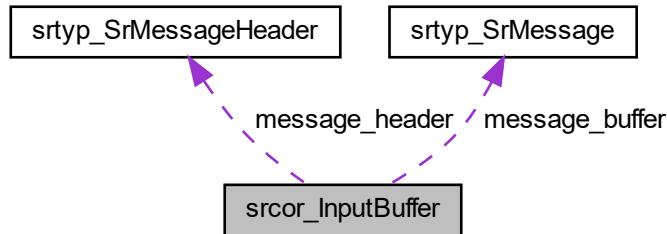
- [sraty_sr_api_types.h](#)

5.6 `srcor_InputBuffer` Struct Reference

Struct for the newly received message input buffer.

```
#include <srcor_sr_core.h>
```

Collaboration diagram for `srcor_InputBuffer`:



Public Attributes

- `bool message_in_buffer`
Flag which indicates, that a new unprocessed message is in the message input buffer.
- `srtyp_SrMessageHeader message_header`
Message header extracted from message in buffer. Valid range as described in `srtyp_SrMessageHeader`.
- `srtyp_SrMessage message_buffer`
Input buffer for newly received message. Valid range as described in `srtyp_SrMessage`.

5.6.1 Detailed Description

Struct for the newly received message input buffer.

The documentation for this struct was generated from the following file:

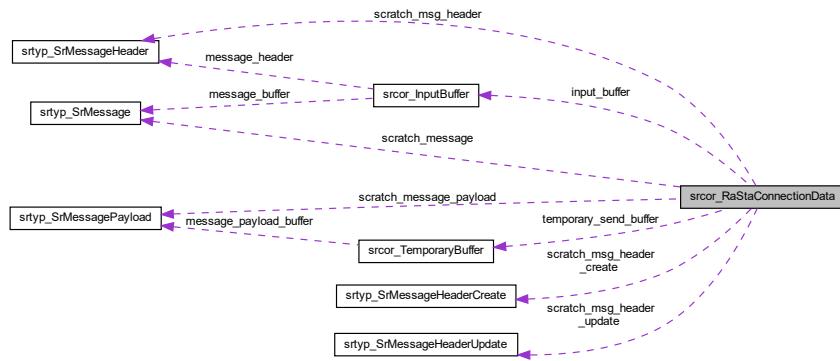
- [srcor_sr_core.h](#)

5.7 srcor_RaStaConnectionData Struct Reference

Struct for the process data of a RaSTA connection.

```
#include <srcor_sr_core.h>
```

Collaboration diagram for srcor_RaStaConnectionData:



Public Attributes

- **uint32_t sequence_number_tx**
SN_T: Sequence number of PDU message to be sent next. The full value range is valid and usable.
- **uint32_t sequence_number_rx**
SN_R: Expected sequence number of the next received PDU message. The full value range is valid and usable.
- **uint32_t confirmed_sequence_number_tx**
Last effective confirmed send sequence number of CS_T. The full value range is valid and usable.
- **uint32_t last_send_confirmed_sequence_number_tx**
Last effective confirmed send sequence number of CS_T. The full value range is valid and usable.
- **uint32_t confirmed_sequence_number_rx**
CS_R: Last received confirmed sequence number. The full value range is valid and usable.
- **uint32_t time_stamp_rx**
TS_R: Time stamp of the time-out related PDU message received last [ms]. The full value range is valid and usable.
- **uint32_t confirmed_time_stamp_rx**
Opposite receive buffer (NsendMax) size [messages]. The full value range is valid and usable.
- **uint32_t time_stamp_tx**
Last time stamp send (for calculation of T_h) [ms]. The full value range is valid and usable.
- **uint16_t detailed_disconnect_reason**
Detailed disconnect reason from the application. The full value range is valid and usable.
- **uint16_t opposite_receive_buffer_size**
Opposite receive buffer (NsendMax) size [messages]. The full value range is valid and usable.
- **bool received_data_pending**
True, if received data is pending on the redundancy channel.
- **srcor_InputBuffer input_buffer**
Input buffer for newly received message. Valid range as described in [srcor_InputBuffer](#).
- **srcor_TemporaryBuffer temporary_send_buffer**
Temporary send buffer for new message payload to send. Valid range as described in [srcor_TemporaryBuffer](#).
- **uint32_t timer_t_i**
Timer T_i [ms]. Dynamically calculated at receipt of time out related messages: T_i = T_max - T_rtd. Valid range: 0 <= value <= T_max.

- `uint32_t t_rtd`
- `uint32_t t_alive`
- `srtyp_SrMessageHeaderCreate scratch_msg_header_create`
Memory used for message creation. Valid range as described in [srtyp_SrMessageHeaderCreate](#).
- `srtyp_SrMessageHeaderUpdate scratch_msg_header_update`
Memory used for message update. Valid range as described in [srtyp_SrMessageHeader](#).
- `srtyp_SrMessageHeader scratch_msg_header`
Memory used for holding header. Valid range as described in [srcor_InputBuffer](#).
- `srtyp_SrMessage scratch_message`
Memory to transfer messages. Valid range as described in [srtyp_SrMessage](#).
- `srtyp_SrMessagePayload scratch_message_payload`
Memory to transfer messages payload. Valid range as described in [srtyp_SrMessagePayload](#).

5.7.1 Detailed Description

Struct for the process data of a RaSTA connection.

5.7.2 Member Data Documentation

5.7.2.1 confirmed_sequence_number_tx `uint32_t srcor_RaStaConnectionData::confirmed_sequence↔_number_tx`

CS_T: Sequence number to be confirmed (which is transmitted at the next PDU message to be sent). The full value range is valid and usable.

5.7.2.2 confirmed_time_stamp_rx `uint32_t srcor_RaStaConnectionData::confirmed_time_stamp_rx`

CTS_R: Confirmed time stamp of the time-out related PDU message received last [ms]. The full value range is valid and usable.

5.7.2.3 t_alive `uint32_t srcor_RaStaConnectionData::t_alive`

Alive time of a message [ms]. Provides a statement to what extend the adaptive channel monitoring has exhausted T_max. Only calculated for time-out relevant messages. The full value range is valid and usable.

5.7.2.4 t_rtd `uint32_t srcor_RaStaConnectionData::t_rtd`

Round trip delay of a message [ms]. Round trip time of a message. Only calculated for time-out relevant messages. Valid range: $0 \leq \text{value} \leq T_{\max}$. The full value range is valid and usable.

The documentation for this struct was generated from the following file:

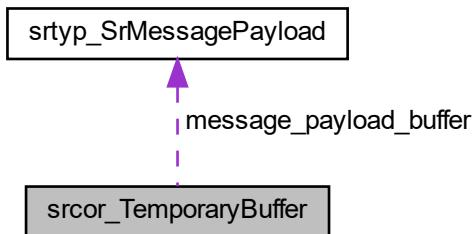
- [srcor_sr_core.h](#)

5.8 `srcor_TemporaryBuffer` Struct Reference

Struct for the message payload temporary buffer.

```
#include <srcor_sr_core.h>
```

Collaboration diagram for `srcor_TemporaryBuffer`:



Public Attributes

- `bool message_in_buffer`
Flag which indicates, that a unprocessed message is in the temporary buffer.
- `srtyp_SrMessagePayload message_payload_buffer`
Buffer for message payload to send. Valid range as described in [srtyp_SrMessagePayload](#).

5.8.1 Detailed Description

Struct for the message payload temporary buffer.

The documentation for this struct was generated from the following file:

- [srcor_sr_core.h](#)

5.9 `srcty_ConnectionConfiguration` Struct Reference

Struct for the configuration of a RaSTA connection.

```
#include <srcty_sr_config_types.h>
```

Public Attributes

- `uint32_t connection_id`
Id of the connection.
- `uint32_t sender_id`
Id of the sender.
- `uint32_t receiver_id`
Id of the receiver.

5.9.1 Detailed Description

Struct for the configuration of a RaSTA connection.

This struct contains the configuration of a RaSTA connection, describing the relation between sender id, receiver id & connection id. Sender id and receiver id must be different from each other.

Implements Requirements [RASW-423](#) Struct Connection Configuration Structure

5.9.2 Member Data Documentation

5.9.2.1 `connection_id` `uint32_t srcty_ConnectionConfiguration::connection_id`

Id of the connection.

Valid range: `0 <= value < srcty_SafetyRetransmissionConfiguration::number_of_connections`

Implements Requirements [RASW-426](#) Connection Id

5.9.2.2 `receiver_id` `uint32_t srcty_ConnectionConfiguration::receiver_id`

Id of the receiver.

The full value range is valid and usable.

Implements Requirements [RASW-435](#) Receiver Id

5.9.2.3 `sender_id` `uint32_t srcty_ConnectionConfiguration::sender_id`

Id of the sender.

The full value range is valid and usable.

Implements Requirements [RASW-425](#) Sender Id

The documentation for this struct was generated from the following file:

- `srcty_sr_config_types.h`

5.10 srcty_Md4InitValue Struct Reference

Struct for the MD4 initial value for a RaSTA network.

```
#include <srcty_sr_config_types.h>
```

Public Attributes

- `uint32_t init_a`
Initial value A. Full value range is valid and usable.
- `uint32_t init_b`
Initial value B. Full value range is valid and usable.
- `uint32_t init_c`
Initial value C. Full value range is valid and usable.
- `uint32_t init_d`
Initial value D. Full value range is valid and usable.

5.10.1 Detailed Description

Struct for the MD4 initial value for a RaSTA network.

This struct contains the 4 initial values for the MD4 safety code. For every single initial value A-D, full value range is valid and usable.

Implements Requirements [RASW-437](#) Struct MD4 Initial Value Structure

[RASW-432](#) Init Value A, B, C, D

The documentation for this struct was generated from the following file:

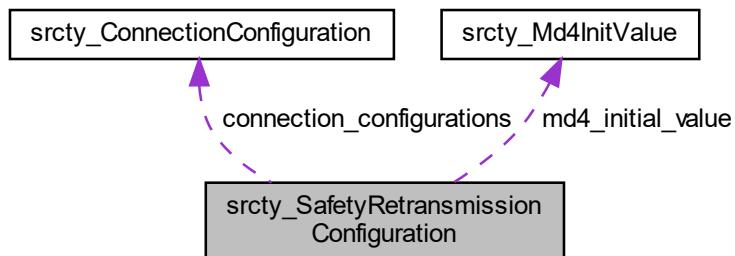
- [srcty_sr_config_types.h](#)

5.11 srcty_SafetyRetransmissionConfiguration Struct Reference

Struct for the configuration data of the SafRetL.

```
#include <srcty_sr_config_types.h>
```

Collaboration diagram for srcty_SafetyRetransmissionConfiguration:



Public Attributes

- `uint32_t rasta_network_id`
RaSTA network id.
- `uint32_t t_max`
Max. accepted age of a message [ms].
- `uint32_t t_h`
Heartbeat period [ms].
- `srcty_SafetyCodeType safety_code_type`
Type of safety code [enum].
- `uint16_t m_w_a`
Maximum number of received, unconfirmed messages [messages].
- `uint16_t n_send_max`
Maximum number of send messages without receiving a confirmation [messages].
- `uint32_t n_max_packet`
Packetization factor.
- `uint32_t n_diag_window`
Diagnosis window size [messages].
- `uint32_t number_of_connections`
Number of configured connections.
- `srcty_ConnectionConfiguration connection_configurations [RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS]`
Configurations of connections.
- `srcty_Md4InitValue md4_initial_value`
MD4 initial value.
- `uint32_t diag_timing_distr_intervals [RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_ARRAY_SIZE]`
Intervals for timing diagnostics [ms].

5.11.1 Detailed Description

Struct for the configuration data of the SafRetL.

Implements Requirements [RASW-427](#) Struct SafetyRetransmissionConfiguration Structure

5.11.2 Member Data Documentation

5.11.2.1 connection_configurations `srcty_ConnectionConfiguration srcty_SafetyRetransmission::connection_configurations [RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS]`

Configurations of connections.

This array of structs contains the configuration of the connections used in the SafRetL. The `srcty_ConnectionConfiguration::connection_configurations` must be in ascending order starting from 0 for the entries in the array. All ranges for the individual structs are described in `srcty_ConnectionConfiguration`.

Implements Requirements [RASW-431](#) Connection Configurations

5.11.2.2 diag_timing_distr_intervals `uint32_t srcty_SafetyRetransmissionConfiguration::diag_←
timing_distr_intervals[RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_ARRAY_SIZE]`

Intervals for timing diagnostics [ms].

This array defines the diagnostic timing intervals. It contains `RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_ARRAY_SIZE` timings which split the maximum time `T_max` in `RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_INTERVALS` time intervals.

- each timing describes its relative starting time from 0
- timings must be arranged in ascending order
- $0 < T1 < T2 < T3 < T4 < t_{max}$

Intervals are defined as follows:

- interval 1: $0 \leq value \leq T1$
- interval 2: $T1 < value \leq T2$
- interval 3: $T2 < value \leq T3$
- interval 4: $T3 < value \leq T4$
- interval 5: $T4 < value \leq t_{max}$

Valid range for every element: $0 < value < \text{srcty_SafetyRetransmissionConfiguration::t_max}$.

Implements Requirements [RASW-433](#) Diagnostic Timing Interval

5.11.2.3 m_w_a `uint16_t srcty_SafetyRetransmissionConfiguration::m_w_a`

Maximum number of received, unconfirmed messages [messages].

Defines the maximum number of received and unconfirmed messages that are allowed. Valid range of `srcty_kMinMWA` \leq value \leq `srcty_kMaxMWA`.

Implements Requirements [RASW-442](#) MWA

5.11.2.4 md4_initial_value `srcty_Md4InitValue srcty_SafetyRetransmissionConfiguration::md4_←
initial_value`

MD4 initial value.

This struct contains the initial values for the MD4 safety code. All ranges for the individual initial values are described in `srcty_Md4InitValue`.

Implements Requirements [RASW-428](#) MD4 Initial Value

5.11.2.5 n_diag_window `uint32_t srcty_SafetyRetransmissionConfiguration::n_diag_window`

Diagnosis window size [messages].

Defines the size of the measurement window for the channel quality measurements. Valid range of `srcty_kMinNDiagWindow` <= value <= `srcty_kMaxNDiagWindow`.

Implements Requirements [RASW-438](#) N diagWindow

5.11.2.6 n_max_packet `uint32_t srcty_SafetyRetransmissionConfiguration::n_max_packet`

Packetization factor.

The maximal packetization factor defines how many messages from one application may at maximum be combined to form aSafRetL message. Valid range: value = `srcty_KNMaxPacket`.

Implements Requirements [RASW-440](#) N maxPacket

5.11.2.7 n_send_max `uint16_t srcty_SafetyRetransmissionConfiguration::n_send_max`

Maximum number of send messages without receiving a confirmation [messages].

Defines the maximum number of messages which the communication party may send without confirmation. Valid range of `srcty_kMinNSendMax` <= value <= `RADEF_MAX_N_SEND_MAX`.

Implements Requirements [RASW-441](#) N sendmax

5.11.2.8 number_of_connections `uint32_t srcty_SafetyRetransmissionConfiguration::number_of_connections`

Number of configured connections.

Defines the number of configured connections that are allowed. Valid range of `srcty_kMinNumberOfRaStaConnections` <= value <= `RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS`.

Implements Requirements [RASW-436](#) Number of Connections

5.11.2.9 rasta_network_id uint32_t srcty_SafetyRetransmissionConfiguration::rasta_network_id

RaSTA network id.

Represent a unique identification which represents the RaSTA network. It is not used by the SW, since only one RaSTA network is supported. The full value range is valid and usable.

Implements Requirements RASW-446 RaSTA Network Id

5.11.2.10 safety_code_type srcty_SafetyCodeType srcty_SafetyRetransmissionConfiguration::safety_code_type

Type of safety code [enum].

Defines the used safety code type of the SafRetL. Valid range: srcty_kSafetyCodeTypeMin <= value < srcty_kSafetyCodeTypeMax

Implements Requirements RASW-443 Safety Code Type

5.11.2.11 t_h uint32_t srcty_SafetyRetransmissionConfiguration::t_h

Heartbeat period [ms].

When the heartbeat period passes without sending any new message, a new heartbeat message must be send so indicate the opposite side that the connection is still alive. Valid range of srcty_kMinTHeartbeat <= value <= srcty_kMaxTHeartbeat.

Implements Requirements RASW-444 T h

5.11.2.12 t_max uint32_t srcty_SafetyRetransmissionConfiguration::t_max

Max. accepted age of a message [ms].

Maximum accepted age of a message before a message timeout is triggered. Valid range of srcty_kMinTMax <= value <= srcty_kMaxTMax.

Implements Requirements RASW-445 T max

The documentation for this struct was generated from the following file:

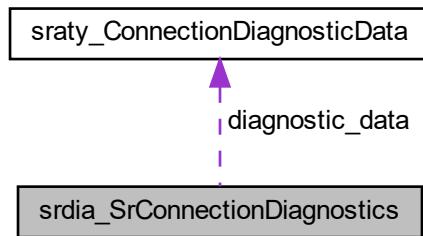
- srcty_sr_config_types.h

5.12 srdia_SrConnectionDiagnostics Struct Reference

Struct for the collection of diagnostic data of a RaSTA connection.

```
#include <srdia_sr_diagnostics.h>
```

Collaboration diagram for srdia_SrConnectionDiagnostics:



Public Attributes

- `uint32_t message_counter`
number of received messages
- `sratty_ConnectionDiagnosticData diagnostic_data`
diagnostic data struct

5.12.1 Detailed Description

Struct for the collection of diagnostic data of a RaSTA connection.

The documentation for this struct was generated from the following file:

- [srdia_sr_diagnostics.h](#)

5.13 srmd4_Md4 Struct Reference

Typedef for MD4 result.

```
#include <srmd4_sr_md4.h>
```

Public Attributes

- `uint8_t md4 [RADEF_MAX_SR_LAYER_SAFETY_CODE_SIZE]`
Array containing the full MD4 hash.

5.13.1 Detailed Description

Typedef for MD4 result.

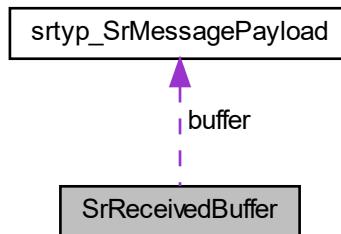
The documentation for this struct was generated from the following file:

- [srmd4_sr_md4.h](#)

5.14 SrReceivedBuffer Struct Reference

Struct for SafRetL received messages payload buffer.

Collaboration diagram for SrReceivedBuffer:



Public Attributes

- `uint16_t read_idx`
buffer read index (next message to read)
- `uint16_t write_idx`
buffer write index (next message to write)
- `uint16_t used_elements`
current amount of used elements in the buffer
- `srtyp_SrMessagePayload buffer [RADEF_MAX_N_SEND_MAX]`
buffer with the payload of correctly received messages, waiting for the read from the application layer

5.14.1 Detailed Description

Struct for SafRetL received messages payload buffer.

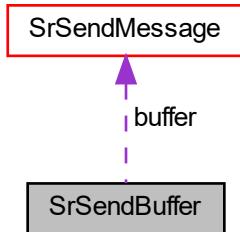
The documentation for this struct was generated from the following file:

- [srrece_sr_received_buffer.c](#)

5.15 SrSendBuffer Struct Reference

Struct for SafRetL send messages buffer.

Collaboration diagram for SrSendBuffer:



Public Attributes

- **uint16_t remove_idx**
buffer remove index (next confirmed message to remove from the buffer)
 - **uint16_t read_idx**
buffer read index (next message to transmit or re-transmit)
 - **uint16_t write_idx**
buffer write index (next new message to write to the buffer)
 - **uint16_t used_elements**
current amount of elements in the buffer
 - **uint16_t not_sent_elements**
current amount of not sent elements in the buffer
 - **SrSendMessage buffer [RADEF_SEND_BUFFER_SIZE]**
buffer with the messages, waiting for to be send

5.15.1 Detailed Description

Struct for SafRetL send messages buffer.

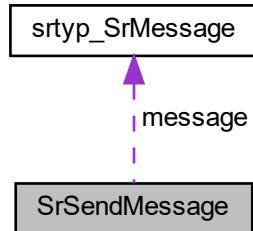
The documentation for this struct was generated from the following file:

- `srsend sr send buffer.c`

5.16 SrSendMessage Struct Reference

Struct for SafRetL send message.

Collaboration diagram for SrSendMessage:



Public Attributes

- bool **already_sent**
was this message already sent (used for retransmission)
- **srtyp_SrMessage message**
message

5.16.1 Detailed Description

Struct for SafRetL send message.

The documentation for this struct was generated from the following file:

- [srsend_sr_send_buffer.c](#)

5.17 srtyp_ProtocolVersion Struct Reference

Typedef for RaSTA protocol version array.

```
#include <srtyp_sr_types.h>
```

Public Attributes

- uint8_t **version** [SRTYP_PROTOCOL_VERSION_SIZE]

5.17.1 Detailed Description

Typedef for RaSTA protocol version array.

5.17.2 Member Data Documentation

5.17.2.1 **version** uint8_t srtyp_ProtocolVersion::version[SRTYP_PROTOCOL_VERSION_SIZE]

Array containing the version in ASCII encoding. The first two bytes represent the major version, the second two bytes represent the minor version. Valid range for every char: `srcty_kProtocolVersionMinValue` <= value <= `srcty_kProtocolVersionMaxValue`.

The documentation for this struct was generated from the following file:

- [srtyp_sr_types.h](#)

5.18 srtyp_SrMessage Struct Reference

Typedef for a SafRetL PDU message.

```
#include <srtyp_sr_types.h>
```

Public Attributes

- **uint16_t message_size**
Used message size [bytes]. Valid range: `RADEF_SR_LAYER_MESSAGE_HEADER_SIZE` <= value <= `RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE`.
- **uint8_t message [RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE]**
Message buffer. For the message data the full value range is valid and usable.

5.18.1 Detailed Description

Typedef for a SafRetL PDU message.

The documentation for this struct was generated from the following file:

- [srtyp_sr_types.h](#)

5.19 srtyp_SrMessageHeader Struct Reference

Typedef for a SafRetL PDU message header.

```
#include <srtyp_sr_types.h>
```

Public Attributes

- **uint16_t message_length**
Message length [bytes]. Valid range: RADEF_SR_LAYER_MESSAGE_HEADER_SIZE <= value <= RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE.
- **srtyp_SrMessageType message_type**
Message type. All enum entries of srtyp_SrMessageType are valid and usable.
- **uint32_t receiver_id**
Receiver identification. The full value range is valid and usable.
- **uint32_t sender_id**
Sender identification. The full value range is valid and usable.
- **uint32_t sequence_number**
SN_PDU: Sequence number. The full value range is valid and usable.
- **uint32_t confirmed_sequence_number**
CS_PDU: Confirmed sequence number. The full value range is valid and usable.
- **uint32_t time_stamp**
TS_PDU: Time stamp [ms]. The full value range is valid and usable.
- **uint32_t confirmed_time_stamp**
CTS_PDU: Confirmed time stamp [ms]. The full value range is valid and usable.

5.19.1 Detailed Description

Typedef for a SafRetL PDU message header.

The documentation for this struct was generated from the following file:

- [srtyp_sr_types.h](#)

5.20 srtyp_SrMessageHeaderCreate Struct Reference

Typedef for SafRetL PDU message header data parameters for creating a message.

```
#include <srtyp_sr_types.h>
```

Public Attributes

- **uint32_t receiver_id**
Receiver identification. The full value range is valid and usable.
- **uint32_t sender_id**
Sender identification. The full value range is valid and usable.
- **uint32_t sequence_number**
SN_PDU: Sequence number. The full value range is valid and usable.
- **uint32_t confirmed_time_stamp**
CTS_PDU: Confirmed time stamp [ms]. The full value range is valid and usable.

5.20.1 Detailed Description

Typedef for SafRetL PDU message header data parameters for creating a message.

The documentation for this struct was generated from the following file:

- [srtyp_sr_types.h](#)

5.21 srtyp_SrMessageHeaderUpdate Struct Reference

Typedef for SafRetL PDU message header data parameters for updating a message header before sending the message.

```
#include <srtyp_sr_types.h>
```

Public Attributes

- **uint32_t confirmed_sequence_number**
CS_PDU: Confirmed sequence number. The full value range is valid and usable.
- **uint32_t time_stamp**
TS_PDU: Time stamp [ms]. The full value range is valid and usable.

5.21.1 Detailed Description

Typedef for SafRetL PDU message header data parameters for updating a message header before sending the message.

The documentation for this struct was generated from the following file:

- [srtyp_sr_types.h](#)

5.22 srtyp_SrMessagePayload Struct Reference

Typedef for a SafRetL PDU message payload.

```
#include <srtyp_sr_types.h>
```

Public Attributes

- **uint16_t payload_size**
Used payload size [bytes]. Valid range: `srcty_kMinSrLayerPayloadDataSize` <= value <= `RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE`.
- **uint8_t payload [RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE]**
Payload buffer. For the message payload the full value range is valid and usable.

5.22.1 Detailed Description

Typedef for a SafRetL PDU message payload.

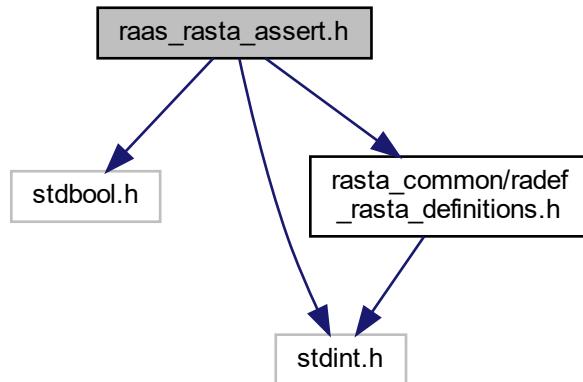
The documentation for this struct was generated from the following file:

- [srtyp_sr_types.h](#)

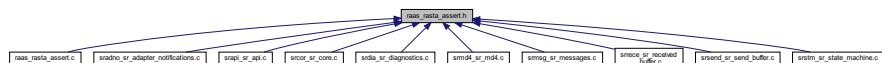
6 File Documentation

6.1 raas_rasta_assert.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
Include dependency graph for raas_rasta_assert.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- void `raas_AssertNotNull` (const void *const pointer, const `radef_RaStaReturnCode` error_reason)
Assert if a pointer is not NULL.
- void `raas_AssertTrue` (const bool condition, const `radef_RaStaReturnCode` error_reason)
Assert a bool condition is true.
- void `raas_AssertU8InRange` (const uint8_t value, const uint8_t min_value, const uint8_t max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a uint8_t value is in a defined range.
- void `raas_AssertU16InRange` (const uint16_t value, const uint16_t min_value, const uint16_t max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a uint16_t value is in a defined range.
- void `raas_AssertU32InRange` (const uint32_t value, const uint32_t min_value, const uint32_t max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a uint32_t value is in a defined range.

6.1.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

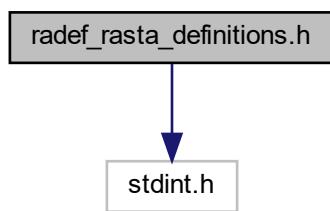
64779fea6efa1199e3a82cbff64181dea3877e8d

Changelog

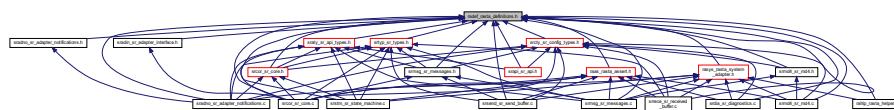
-- Initial version (-, -)

6.2 `radef_rasta_definitions.h` File Reference

```
#include <stdint.h>
Include dependency graph for radef_rasta_definitions.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `radef_TransportChannelDiagnosticData`

Struct for the diagnostic data from a transport channel.

Macros

- **#define PRIVATE static**
Macro for local variables which shall be accessible during unit tests.
 - **#define RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS (2U)**
Maximum number of RaSTA connections per RaSTA network.
 - **#define RADEF_MAX_SR_LAYER_PAYLOAD_DATA_SIZE (1055U)**
Maximum payload size of a SafRetL PDU message [Bytes].
 - **#define RADEF_SR_LAYER_MESSAGE_HEADER_SIZE (28U)**
Header size of a SafRetL PDU message [Bytes].
 - **#define RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE (2U)**
Application message length size of a SafRetL PDU message [Bytes].
 - **#define RADEF_MAX_SR_LAYER_SAFETY_CODE_SIZE (16U)**
Maximum safety code size of a SafRetL PDU message [Bytes].
 - **#define RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE**
Maximum size of a SafRetL PDU message (including header and safety code) [Bytes].
 - **#define RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_INTERVALS (5U)**
Number of received message timing distribution diagnostic intervals.
 - **#define RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_ARRAY_SIZE (RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_INTERVALS - 1U)**
Size of timing distribution diagnostic interval array. Contains one element less than intervals since last element is set to t_max.
 - **#define RADEF_MAX_N_SEND_MAX (20U)**
Maximum number of entries in the received buffer [messages].
 - **#define RADEF_SEND_BUFFER_SIZE (RADEF_MAX_N_SEND_MAX)**
Defines the number of send buffer entries [messages].
 - **#define RADEF_MAX_NUMBER_OF_RED_CHANNELS (RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS)**
Maximum number of redundancy channels.
 - **#define RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS (2U)**
Maximum number of transport channels per redundancy channel.
 - **#define RADEF_RED_LAYER_MESSAGE_HEADER_SIZE (8U)**
Header size of a RedL PDU message [Bytes].
 - **#define RADEF_MAX_RED_LAYER_CHECK_CODE_SIZE (4U)**
Maximum check code size of a RedL PDU message [Bytes].
 - **#define RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE (RADEF_RED_LAYER_MESSAGE_HEADER_SIZE + RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE + RADEF_MAX_RED_LAYER_CHECK_CODE_SIZE)**
Maximum size of RedL PDU message (including RedL header, max. SafRetL PDU message size and max. check code size) [Bytes].

- #define **RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE** (**RADEF_RED_LAYER_MESSAGE_HEADER_SIZE** + **RADEF_SR_LAYER_MESSAGE_HEADER_SIZE**)
Minimum size of RedL PDU message (including RedL header, min. SafRetL PDU message size (only SafRetL header) and min. check code size (none)) [Bytes].
- #define **RADEF_MAX_DEFER_QUEUE_SIZE** (10U)
Maximum size of a redundancy channel defer queue [messages].
- #define **RADEF_MAX_RED_LAYER_N_DIAGNOSIS** (1000U)
Maximum RedL diagnosis window size (Ndiagnosis) [messages].

Enumerations

- enum **radef_RaStaReturnCode** {
 radef_kMin = 0 , **radef_kNoError** = 0 , **radef_kNoMessageReceived** = 1 , **radef_kNoMessageToSend** = 2 ,
 radef_kNotInitialized = 3 , **radef_kAlreadyInitialized** = 4 , **radef_kInvalidConfiguration** = 5 , **radef_kInvalidParameter**
 = 6 ,
 radef_kInvalidMessageType = 7 , **radef_kInvalidMessageSize** = 8 , **radef_kInvalidBufferSize** = 9 ,
 radef_kInvalidMessageCrc = 10 ,
 radef_kInvalidMessageMd4 = 11 , **radef_kReceiveBufferFull** = 12 , **radef_kDeferQueueEmpty** = 13 ,
 radef_kSendBufferFull = 14 ,
 radef_kInvalidSequenceNumber = 15 , **radef_kInternalError** = 16 , **radef_kInvalidOperationInCurrentState** =
 17 , **radef_kMax** }

Enum for function return codes of the RaSTA stack.

6.2.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn
Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

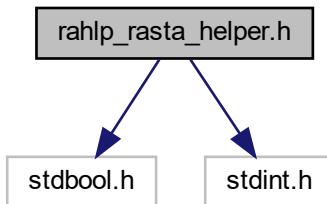
64779fea6efa1199e3a82cbff64181dea3877e8d

Changelog :: Initial version (-, -)

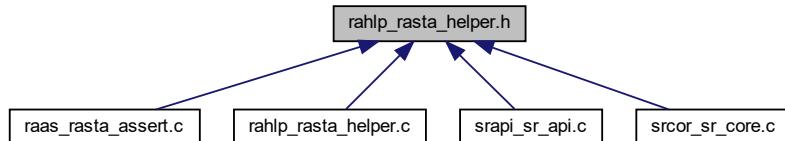
6.3 rahlp_rasta_helper.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for rahlp_rasta_helper.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool [rahlp_IsU16InRange](#) (const uint16_t value, const uint16_t min_value, const uint16_t max_value)
Checks if a uint16_t value is in a defined range.
- bool [rahlp_IsU32InRange](#) (const uint32_t value, const uint32_t min_value, const uint32_t max_value)
Checks if a uint32_t value is in a defined range.

6.3.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

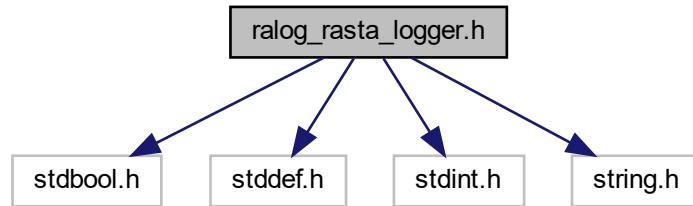
Version

64779fea6efa1199e3a82cbff64181dea3877e8d

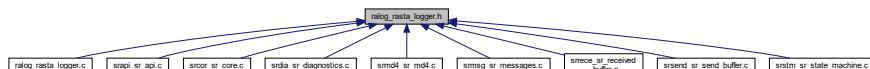
[Changelog](#) :: Initial version (-, -)

6.4 ralog_rasta_logger.h File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
Include dependency graph for ralog_rasta_logger.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- `#define ralog_ENABLE_LOGGER 0`
Global logger disable.
- `#define ralog_INIT_LOGGER(log_level) 0U`
Empty placeholder macro for `ralog_INIT_LOGGER()` used for RELEASE build.
- `#define ralog_LOG_ERROR(logger_id, message, ...)`
Empty placeholder macro for `ralog_LOG_ERROR()` used for RELEASE build.
- `#define ralog_LOG_WARN(logger_id, message, ...)`
Empty placeholder macro for `ralog_LOG_WARN()` used for RELEASE build.
- `#define ralog_LOG_INFO(logger_id, message, ...)`
Empty placeholder macro for `ralog_LOG_INFO()` used for RELEASE build.
- `#define ralog_LOG_DEBUG(logger_id, message, ...)`
Empty placeholder macro for `ralog_LOG_DEBUG()` used for RELEASE build.

6.4.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

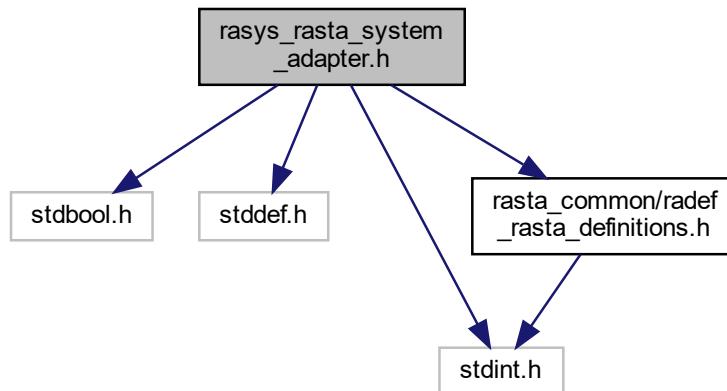
Version

64779fea6efa1199e3a82cbff64181dea3877e8d

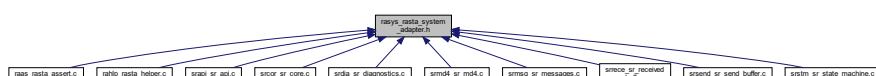
Changelog :: Initial version (-, -)

6.5 rasys_rasta_system_adapter.h File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
Include dependency graph for rasys_rasta_system_adapter.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- `uint32_t rasys_GetTimerValue (void)`

Returns the actual value of a free running up counting timer.
- `uint32_t rasys_GetTimerGranularity (void)`

Returns the granularity of the free running up counting timer.
- `uint32_t rasys_GetRandomNumber (void)`

Returns a random generated number within the uint32_t type range.
- `void rasys_FatalError (const radef_RaStaReturnCode error_reason)`

Fatal error function.

6.5.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

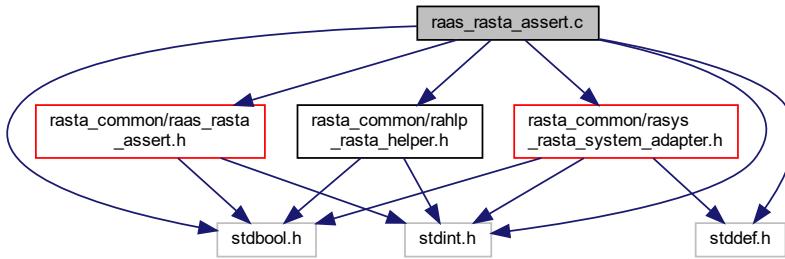
64779fea6efa1199e3a82cbff64181dea3877e8d

Changelog :: Initial version (-, -)

6.6 raas_rasta_assert.c File Reference

Implementation of RaSTA assert functions.

```
#include "rasta_common/raas_rasta_assert.h"
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_common/rahlp_rasta_helper.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
Include dependency graph for raas_rasta_assert.c:
```



Functions

- void `raas_AssertNotNull` (const void *const pointer, const `radef_RaStaReturnCode` error_reason)
Assert if a pointer is not NULL.
- void `raas_AssertTrue` (const bool condition, const `radef_RaStaReturnCode` error_reason)
Assert a bool condition is true.
- void `raas_AssertU8InRange` (const `uint8_t` value, const `uint8_t` min_value, const `uint8_t` max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a `uint8_t` value is in a defined range.
- void `raas_AssertU16InRange` (const `uint16_t` value, const `uint16_t` min_value, const `uint16_t` max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a `uint16_t` value is in a defined range.
- void `raas_AssertU32InRange` (const `uint32_t` value, const `uint32_t` min_value, const `uint32_t` max_value, const `radef_RaStaReturnCode` error_reason)
Assert if a `uint32_t` value is in a defined range.

6.6.1 Detailed Description

Implementation of RaSTA assert functions.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

64779fea6efa1199e3a82cbff64181dea3877e8d

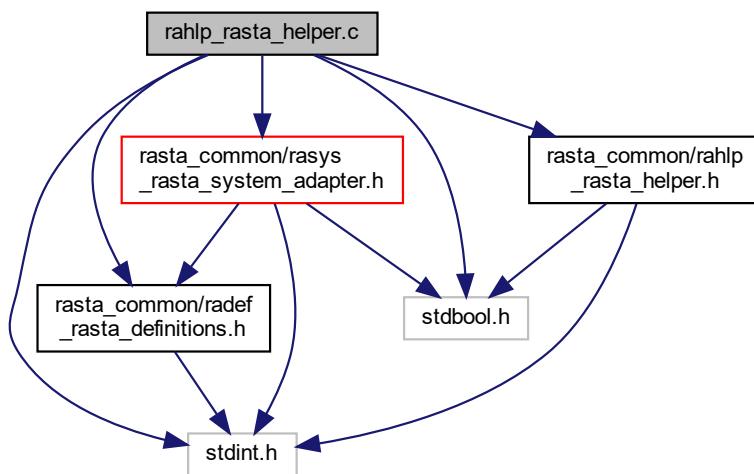
Changelog :: Initial version (-, -)

6.7 rahlp_rasta_helper.c File Reference

Implementation of RaSTA helper functions.

```
#include "rasta_common/rahlp_rasta_helper.h"
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
```

Include dependency graph for rahlp_rasta_helper.c:



Functions

- bool `rahlp_IsU16InRange` (const uint16_t value, const uint16_t min_value, const uint16_t max_value)
Checks if a uint16_t value is in a defined range.
- bool `rahlp_IsU32InRange` (const uint32_t value, const uint32_t min_value, const uint32_t max_value)
Checks if a uint32_t value is in a defined range.

6.7.1 Detailed Description

Implementation of RaSTA helper functions.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

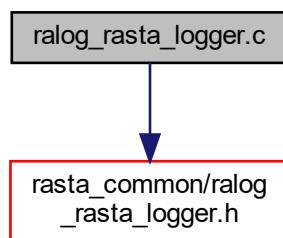
64779fea6efa1199e3a82cbff64181dea3877e8d

Changelog :: Initial version (-, -)

6.8 ralog_rasta_logger.c File Reference

Implementation of RaSTA debug logger.

```
#include "rasta_common/ralog_rasta_logger.h"  
Include dependency graph for ralog_rasta_logger.c:
```



6.8.1 Detailed Description

Implementation of RaSTA debug logger.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

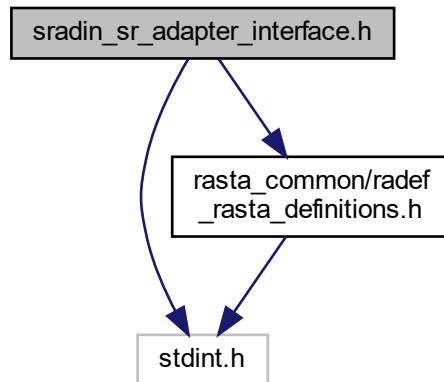
Version

64779fea6efa1199e3a82cbff64181dea3877e8d

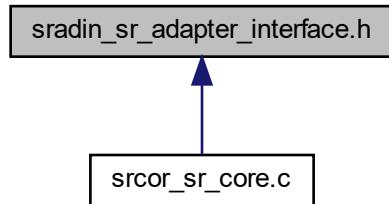
Changelog :: Initial version (-, -)

6.9 sradin_sr_adapter_interface.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
Include dependency graph for sradin_sr_adapter_interface.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- void [sradin_Init](#) (void)
Initialize SaFRetL adapter.
- void [sradin_OpenRedundancyChannel](#) (const uint32_t redundancy_channel_id)
Open a redundancy channel.
- void [sradin_CloseRedundancyChannel](#) (const uint32_t redundancy_channel_id)
Close a redundancy channel.
- void [sradin_SendMessage](#) (const uint32_t redundancy_channel_id, const uint16_t message_size, const uint8_t *const message_data)
Send a message over a redundancy channel.
- [radef_RaStaReturnCode](#) [sradin_ReadMessage](#) (const uint32_t redundancy_channel_id, const uint16_t buffer_size, uint16_t *const message_size, uint8_t *const message_buffer)
Read a received message from a redundancy channel.

6.9.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

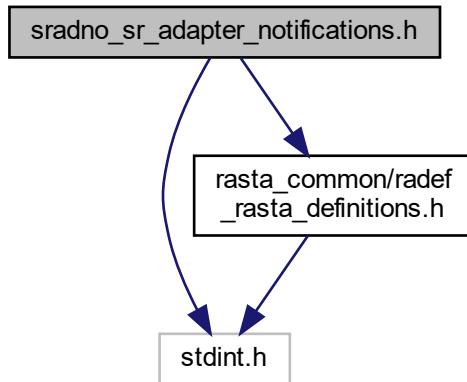
Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

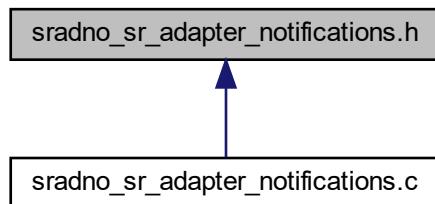
[Changelog](#) :: Initial version (-, -)

6.10 sradno_sr_adapter_notifications.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
Include dependency graph for sradno_sr_adapter_notifications.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- `radef_RaStaReturnCode sradno_MessageReceivedNotification (const uint32_t red_channel_id)`
Message received notification from the RedL.
- `radef_RaStaReturnCode sradno_DiagnosticNotification (const uint32_t red_channel_id, const uint32_t tr_channel_id, const radef_TransportChannelDiagnosticData tr_channel_diagnostic_data)`
Diagnostic notification from the RedL.

6.10.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

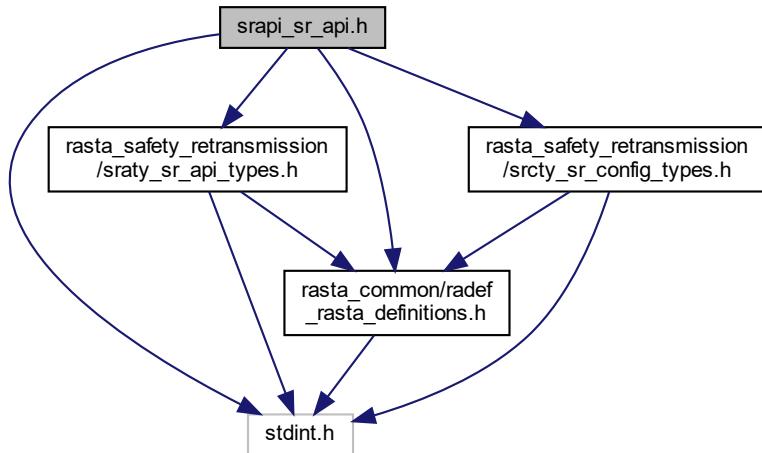
3d5012751f7f8e2a1c005ab71aeb88c68eaf7107

Changelog :: Initial version (-, -)

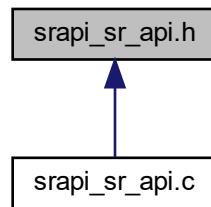
SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4932: sr_adapter_notification module functions are not error tolerant (08.12.2022, N. Andres)

6.11 srapi_sr_api.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
Include dependency graph for srapi_sr_api.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- `raDef_RaStaReturnCode sraPI_Init (const srcty_SafetyRetransmissionConfiguration *const safety_retransmission_configuration)`
Initialize SafRetL.
- `raDef_RaStaReturnCode sraPI_GetInitializationState (void)`
Get the initialization state of the SafRetL.
- `raDef_RaStaReturnCode sraPI_OpenConnection (const uint32_t sender_id, const uint32_t receiver_id, const uint32_t network_id, uint32_t *const connection_id)`
Open a RaSTA connection.
- `raDef_RaStaReturnCode sraPI_CloseConnection (const uint32_t connection_id, const uint16_t detailed_reason)`
Close a RaSTA connection.
- `raDef_RaStaReturnCode sraPI_SendData (const uint32_t connection_id, const uint16_t message_size, const uint8_t *const message_data)`
Send a RaSTA data message.
- `raDef_RaStaReturnCode sraPI_ReadData (const uint32_t connection_id, const uint16_t buffer_size, uint16_t *const message_size, uint8_t *const message_buffer)`
Read the data of a received RaSTA message.
- `raDef_RaStaReturnCode sraPI_GetConnectionState (const uint32_t connection_id, sraty_ConnectionStates *const connection_state, sraty_BufferUtilisation *const buffer_utilisation, uint16_t *const opposite_buffer_size)`
Get the state of a connection.
- `raDef_RaStaReturnCode sraPI_CheckTimings (void)`
Check SafRetL timings.

6.11.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

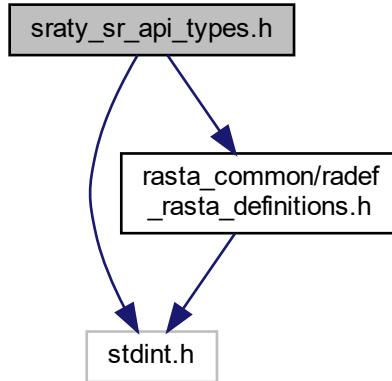
Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

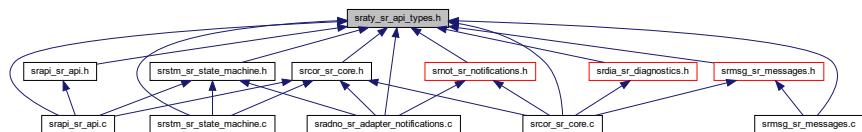
Changelog :: Initial version (-, -)

6.12 sraty_sr_api_types.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
Include dependency graph for sraty_sr_api_types.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct **sraty_BufferUtilisation**
Struct for the buffer utilisation of the SaFRetL buffers.
- struct **sraty_ConnectionDiagnosticData**
Struct for the diagnostic data of a RaSTA connection.
- struct **sraty_RedundancyChannelDiagnosticData**
Struct for the diagnostic data from a redundancy channel.

Enumerations

- enum **sraty_ConnectionStates** {
 sraty_kConnectionMin = 0, **sraty_kConnectionNotInitialized** = 0, **sraty_kConnectionClosed**, **sraty_kConnectionDown**,
 , **sraty_kConnectionStart**, **sraty_kConnectionUp**, **sraty_kConnectionRetransRequest**, **sraty_kConnectionRetransRunning**,
 , **sraty_kConnectionMax** }

Enum for the state of a RaSTA connection.

- enum `sraty_DiscReason` {

`sraty_kDiscReasonMin = 0U` , `sraty_kDiscReasonUserRequest = 0U` , `sraty_kDiscReasonNotInUse = 1U` ,

`sraty_kDiscReasonUnexpectedMessage = 2U` ,

`sraty_kDiscReasonSequenceNumberError = 3U` , `sraty_kDiscReasonTimeout = 4U` , `sraty_kDiscReasonServiceNotAllowed = 5U` ,
 `sraty_kDiscReasonProtocolVersionError = 6U` ,

`sraty_kDiscReasonRetransmissionFailed = 7U` , `sraty_kDiscReasonProtocolSequenceError = 8U` ,

`sraty_kDiscReasonMax` }

Enum for disconnect reason.

6.12.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

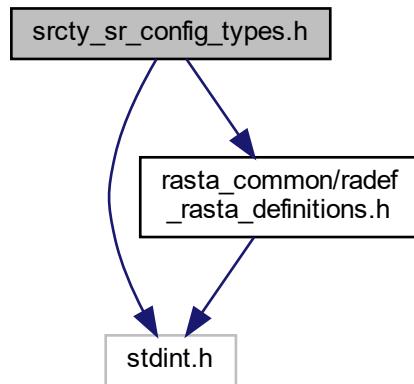
e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog

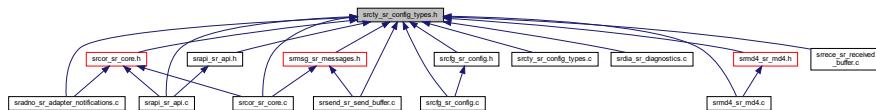
--: Initial version (-, -)

6.13 srcty_sr_config_types.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
Include dependency graph for srcty_sr_config_types.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `srcty_ConnectionConfiguration`
Struct for the configuration of a RaSTA connection.
- struct `srcty_Md4InitValue`
Struct for the MD4 initial value for a RaSTA network.
- struct `srcty_SafetyRetransmissionConfiguration`
Struct for the configuration data of the SafRetL.

Enumerations

- enum `srcty_SafetyCodeType` {
 `srcty_kSafetyCodeTypeMin` = 1 , `srcty_kSafetyCodeTypeNone` = 1 , `srcty_kSafetyCodeTypeLowerMd4` = 2 ,
 `srcty_kSafetyCodeTypeFullMd4` = 3 ,
 `srcty_kSafetyCodeTypeMax` }
- Enum for the safety code type.*

Variables

- const `uint32_t srcty_kMinNumberOfRaStaConnections`
Minimum number of RaSTA connections per RaSTA network.
- const `uint16_t srcty_kMinSrLayerPayloadDataSize`
Minimum SafRetL payload data size.
- const `uint32_t srcty_kMinTMax`
Minimum max. accepted age of a message (Tmax) [ms].
- const `uint32_t srcty_kMaxTMax`
Maximum max. accepted age of a message (Tmax) [ms].
- const `uint32_t srcty_kMinTHearbeat`
Minimum heartbeat period (Th) [ms].
- const `uint32_t srcty_kMaxTHearbeat`
Maximum heartbeat period (Th) [ms].
- const `uint16_t srcty_kMinNSendMax`
Minimum receive buffer size (Nsendmax) [messages].
- const `uint16_t srcty_kMinMWA`
Minimum max. number of received, unconfirmed messages (MWA) [messages].
- const `uint16_t srcty_kMaxMWA`
Maximum max. number of received, unconfirmed messages (MWA) [messages].
- const `uint32_t srcty_kNMaxPacket`
Packetization factor (must always be 1!).
- const `uint32_t srcty_kMinNDiagWindow`
Minimum SafRetL diagnosis window size (Ndiagwindow) [messages].

- const uint32_t **srcty_kMaxNDiagWindow**
Maximum SafRetL diagnosis window size (Ndiagwindow) [messages].
- const uint16_t **srcty_kByteCountUInt16**
Byte count of type UInt16_t [bytes].
- const uint16_t **srcty_kByteCountUInt32**
Byte count of type UInt32_t [bytes].
- const uint16_t **srcty_kByteCountUInt64**
Byte count of type UInt64_t [bytes].
- const uint8_t **srcty_kProtocolVersionMinValue**
Minimum ASCII character value for protocol verion.
- const uint8_t **srcty_kProtocolVersionMaxValue**
Maximum ASCII character value for protocol verion.
- const uint32_t **srcty_kMinFreeEntriesSendBufferForRetr**
Minimum amount of free entries in the send buffer in case of a retransmission.
- const uint32_t **srcty_kMinFreeEntriesReceivedBufferForReceive**
Minimum amount of free entries in the received buffer in case of receiving a message.

6.13.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

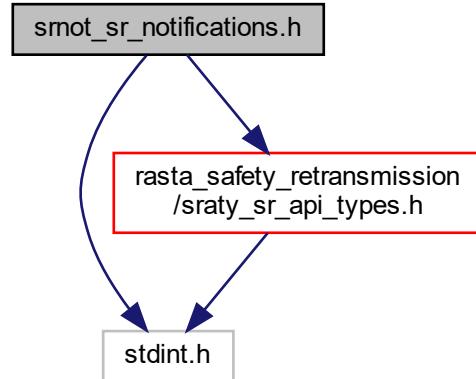
Changelog

-- Initial version (-, -)

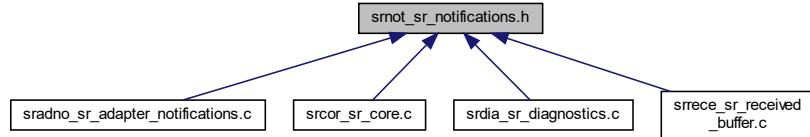
6.14 srnot_sr_notifications.h File Reference

```
#include <stdint.h>
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
```

Include dependency graph for srnot_sr_notifications.h:



This graph shows which files directly or indirectly include this file:



Functions

- void `srnot_MessageReceivedNotification` (const uint32_t connection_id)
SafRetL message received notification function.
- void `srnot_ConnectionStateNotification` (const uint32_t connection_id, const `sraty_ConnectionStates` connection_state, const `sraty_BufferUtilisation` buffer_utilisation, const uint16_t opposite_buffer_size, const `sraty_DiscReason` disconnect_reason, const uint16_t detailed_disconnect_reason)
SafRetL connection state notification function.
- void `srnot_SrDiagnosticNotification` (const uint32_t connection_id, const `sraty_ConnectionDiagnosticData` connection_diagnostic_data)
SafRetL diagnostic data notification function.
- void `srnot_RedDiagnosticNotification` (const uint32_t connection_id, const `sraty_RedundancyChannelDiagnosticData` redundancy_channel_diagnostic_data)
Forwarded RedL diagnostic data notification function.

6.14.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

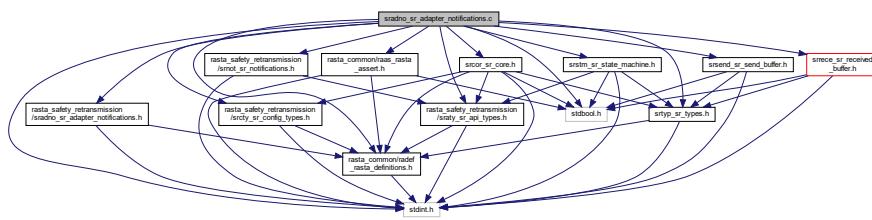
Changelog :: Initial version (-, -)

6.15 sradno_sr_adapter_notifications.c File Reference

Implementation of RaSTA SafRetL adapter notifications.

```
#include "rasta_safety_retransmission/sradno_sr_adapter_notifications.h"
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
#include "rasta_safety_retransmission/srnot_sr_notifications.h"
#include "srcor_sr_core.h"
#include "srrecep_sr_received_buffer.h"
#include "srssend_sr_send_buffer.h"
#include "srstsm_sr_state_machine.h"
#include "srtyp_sr_types.h"
```

Include dependency graph for sradno_sr_adapter_notifications.c:



Functions

- **radef_RaStaReturnCode sradno_MessageReceivedNotification (const uint32_t red_channel_id)**
Message received notification from the RedL.
- **radef_RaStaReturnCode sradno_DiagnosticNotification (const uint32_t red_channel_id, const uint32_t tr_← channel_id, const radef_TransportChannelDiagnosticData tr_channel_diagnostic_data)**
Diagnostic notification from the RedL.

6.15.1 Detailed Description

Implementation of RaSTA SafRetL adapter notifications.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

3d5012751f7f8e2a1c005ab71aeb88c68eaf7107

Changelog

-- Initial version (-, -)

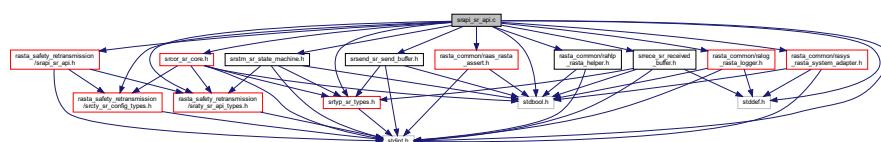
SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4932: sr_adapter_notification module functions are not error tolerant (08.12.2022, N. Andres)

6.16 srapi_sr_api.c File Reference

Implementation of RaSTA SafRetL API.

```
#include "rasta_safety_retransmission/srapi_sr_api.h"
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/rahlp_rasta_helper.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
#include "srcor_sr_core.h"
#include "srrecep_sr_received_buffer.h"
#include "srssend_sr_send_buffer.h"
#include "srstm_sr_state_machine.h"
#include "srtyp_sr_types.h"

Include dependency graph for srapi_sr_api.c:
```



Functions

- `raDef_RaStaReturnCode sraPI_Init (const srcty_SafetyRetransmissionConfiguration *const safety_retransmission_configuration Initialize SafRetL.`
- `raDef_RaStaReturnCode sraPI_GetInitializationState (void) Get the initialization state of the SafRetL.`
- `raDef_RaStaReturnCode sraPI_OpenConnection (const uint32_t sender_id, const uint32_t receiver_id, const uint32_t network_id, uint32_t *const connection_id) Open a RaSTA connection.`
- `raDef_RaStaReturnCode sraPI_CloseConnection (const uint32_t connection_id, const uint16_t detailed_reason) Close a RaSTA connection.`
- `raDef_RaStaReturnCode sraPI_SendData (const uint32_t connection_id, const uint16_t message_size, const uint8_t *const message_data) Send a RaSTA data message.`
- `raDef_RaStaReturnCode sraPI_ReadData (const uint32_t connection_id, const uint16_t buffer_size, uint16_t *const message_size, uint8_t *const message_buffer) Read the data of a received RaSTA message.`
- `raDef_RaStaReturnCode sraPI_GetConnectionState (const uint32_t connection_id, sraty_ConnectionStates *const connection_state, sraty_BufferUtilisation *const buffer_utilisation, uint16_t *const opposite_buffer_size) Get the state of a connection.`
- `raDef_RaStaReturnCode sraPI_CheckTimings (void) Check SafRetL timings.`

Variables

- `PRIVATE bool sraPI_initialized = false Initialization state of the module. true, if the module is initialized.`
- `PRIVATE const srcty_SafetyRetransmissionConfiguration * sraPI_sr_configuration = NULL Pointer to SafRetL configuration.`
- `PRIVATE srty_SrMessagePayload sraPI_scratch_message_payload Memory to transfer messages.`

6.16.1 Detailed Description

Implementation of RaSTA SafRetL API.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog

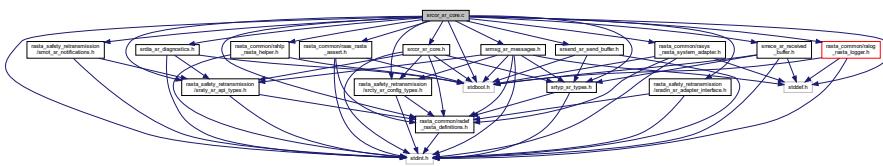
--: Initial version (-, -)

6.17 srcor_sr_core.c File Reference

Implementation of RaSTA SafRetL core module.

```
#include "srcor_sr_core.h"
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_common/rahlp_rasta_helper.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_safety_retransmission/sradin_sr_adapter_interface.h"
#include "rasta_safety_retransmission/srcty_sr_api_types.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
#include "rasta_safety_retransmission/srnot_sr_notifications.h"
#include "srdia_sr_diagnostics.h"
#include "srmsg_sr_messages.h"
#include "srrece_sr_received_buffer.h"
#include "srsend_sr_send_buffer.h"
#include "srtyp_sr_types.h"
```

Include dependency graph for srcor_sr_core.c:



Functions

- static bool **CheckConnectionConfigurations** (const uint32_t number_of_connections, const **srtyp_ConnectionConfiguration** *const connection_configurations)

Check if the connection configurations are valid.
- static bool **IsMessageTimeoutRelated** (const **srtyp_SrMessageType** message_type)

Checks if a message is timeout related or not.
- static uint32_t **GetCurrentSequenceNumberAndIncrementNumber** (const uint32_t connection_id)

Get the Current Sequence Number SN_T and Increment Number.
- static bool **SendPendingMessagesWithFlowControlAllowed** (const uint32_t connection_id)

Checks if the flow control allows to send message to the opposite side.
- static void **ReceivedFlowControlCheck** (const uint32_t connection_id, const **srtyp_SrMessageType** message_type)

Perform the received flow control check.
- static bool **GeneralMessageCheck** (const uint32_t connection_id, const **srtyp_SrMessage** *const msg, **srtyp_SrMessageHeader** *const msg_hdr)

Check the general part of a message.
- static bool **CheckSequenceNumberRange** (const uint32_t connection_id, const **srtyp_SrMessageHeader** *const msg_hdr)

Check the sequence number range of a new received message.
- static bool **CheckSequenceNumber** (const uint32_t connection_id, const **srtyp_SrMessageHeader** *const msg_hdr)

- static bool `CheckConfirmedSequenceNumber` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr)

Check the sequence number of a new received message.
- static bool `CheckTimeStamp` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr)

Check the confirmed sequence number of a new received message.
- static bool `CheckConfirmedTimeStamp` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr)

Checks the time stamp of a new received message.
- static bool `CalculateTimeliness` (const uint32_t connection_id, const `srtyp_SrMessageHeader` *const msg_hdr, const uint32_t current_time)

Checks the confirmed time stamp of a new received message.
- static void `SetConnectionEvent` (const `srtyp_SrMessageType` message_type, `srtyp_ConnectionEvents` *const connection_event)

Calculate the timeliness of a new received message.
- bool `srcor_IsConfigurationValid` (const `srcty_SafetyRetransmissionConfiguration` *const sr_layer_configuration)

Set the connection event according to the message type.
- void `srcor_Init` (const `srcty_SafetyRetransmissionConfiguration` *const sr_layer_configuration)

Checks if a SafRetL configuration is valid.
- `radef_RaStaReturnCode` `srcor_GetConnectionId` (const uint32_t sender_id, const uint32_t receiver_id, uint32_t *const connection_id)

Initialize SafRetL core module.
- Get the connection identification associated with the specified sender and receiver identification.
 - void `srcor_InitRaStaConnData` (const uint32_t connection_id)

Initialize the SafRetL core data of a dedicated RaSTA connection.
 - void `srcor_CloseRedundancyChannel` (const uint32_t connection_id)

Close the redundancy channel of a rasta connection and send a diagnostic notification.
 - void `srcor_ReceiveMessage` (const uint32_t connection_id, `srtyp_ConnectionEvents` *const connection_event, bool *const sequence_number_in_seq, bool *const confirmed_time_stamp_in_seq)

Read and analyze a received SafRetL message.
 - bool `srcor_ProcessReceivedMessage` (const uint32_t connection_id)

Process a successfully received message of a dedicated connection.
 - void `srcor_UpdateConfirmedTxSequenceNumber` (const uint32_t connection_id)

Update confirmed sequence number to transmit from the message in the input buffer.
 - void `srcor_UpdateConfirmedRxSequenceNumber` (const uint32_t connection_id)

Update confirmed sequence number receive from message in input buffer.
 - bool `srcor_IsProtocolVersionAccepted` (const uint32_t connection_id)

Checks if the requested protocol version is accepted.
 - void `srcor_SetReceivedMessagePendingFlag` (const uint32_t connection_id)

Set the message pending flag of a dedicated RaSTA connection.
 - bool `srcor_GetReceivedMessagePendingFlag` (const uint32_t connection_id)

Get the received message pending flag for a dedicated RaSTA connection.
 - void `srcor_WriteMessagePayloadToTemporaryBuffer` (const uint32_t connection_id, const uint16_t message_payload_size, const uint8_t *const message_payload)

Write message payload to temporary buffer for messages to send.
 - void `srcor_ClearInputBufferMessagePendingFlag` (const uint32_t connection_id)

Clear input buffer messages pending flag.
 - void `srcor_SendDataMessage` (const uint32_t connection_id)

Create and send a SafRetL data message from the temporary buffer.
 - void `srcor_SendConnReqMessage` (const uint32_t connection_id)

Create and send a SafRetL connection request message.
 - void `srcor_SendConnRespMessage` (const uint32_t connection_id)

- Create and send a SafRetL connection response message.*
- void `srcor_SendDiscReqMessage` (const uint32_t connection_id, const `sraty_DiscReason` disconnect_reason)
Create and send a SafRetL disconnection request message.
 - void `srcor_SetDiscDetailedReason` (const uint32_t connection_id, const uint16_t detailed_disconnect_reason)
Set detailed reason for disconnection request message.
 - void `srcor_SendHbMessage` (const uint32_t connection_id)
Create and send a SafRetL heartbeat message.
 - void `srcor_SendRetrReqMessage` (const uint32_t connection_id)
Create and send a SafRetL retransmission request message.
 - void `srcor_HandleRetrReq` (const uint32_t connection_id)
Handle a retransmission request and send a SafRetL retransmission response, heartbeat or data message.
 - bool `srcor_IsRetrReqSequenceNumberAvailable` (const uint32_t connection_id)
Checks if a requested retransmission sequence number is available in the send buffer.
 - bool `srcor_IsConnRoleServer` (const uint32_t connection_id)
Returns true, if the own connection role is server.
 - bool `srcor_IsMessageTimeout` (const uint32_t connection_id)
Checks if a message timeout occurred.
 - bool `srcor_IsHeartbeatInterval` (const uint32_t connection_id)
Checks if the heartbeat interval is elapsed.
 - bool `srcor_IsReceivedMsgPendingAndBuffersNotFull` (const uint32_t connection_id)
Check if received messages are pending and send & received buffer are not full.
 - void `srcor_SendPendingMessages` (const uint32_t connection_id)
Send pending messages from the send buffer, if the flow control allows to send.
 - void `srcor_SendConnectionStateNotification` (const uint32_t connection_id, const `sraty_ConnectionStates` connection_state, const `sraty_DiscReason` disconnect_reason)
Send a connection state changed notification to the application layer.
 - void `srcor_GetBufferSizeAndUtilisation` (const uint32_t connection_id, `sraty_BufferUtilisation` *const buffer_utilisation, uint16_t *const opposite_buffer_size)
Get the opposite receive buffer size and the own buffer utilisation for the connection state request.

Variables

- **PRIVATE** bool `srcor_initialized` = false
Initialization state of the module. true, if the module is initialized.
- **PRIVATE** const `srcty_SafetyRetransmissionConfiguration` * `srcor_sr_configuration` = NULL
Pointer to SafRetL configuration.
- **PRIVATE** `srcor_RaStaConnectionData` `srcor_rasta_connections` [`RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS`]
Process data of the rasta connections.
- **PRIVATE** uint32_t `srcor_timer_granularity` = 0U
Granularity of the system timer [ms].
- **PRIVATE** uint16_t `srcor_logger_id`
ID of the sr_core debug logger.
- const `srtyp_ProtocolVersion` `srcor_kProtocolVersion`
Definition of RaSTA protocol version 03.03 (all 4 bytes are decimal digits in ASCII)

6.17.1 Detailed Description

Implementation of RaSTA SafRetL core module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

420e705e9ad1b43bfa1c0be45f837e0fe2573bd4

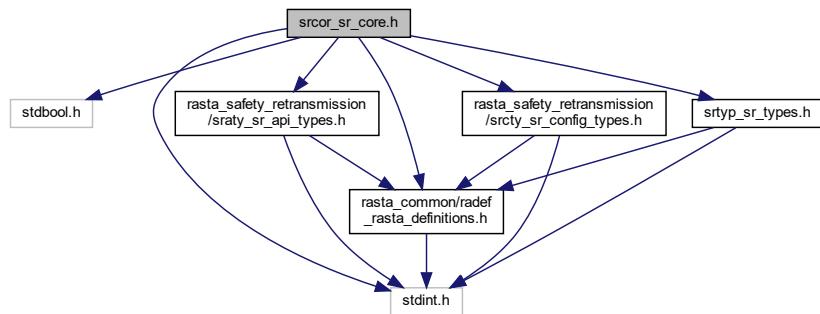
Changelog :: Initial version (-, -)

SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4933: Duplicated sent of diagnostic notification in sr_core when closing the connection (06.12.2022, N. Andres)

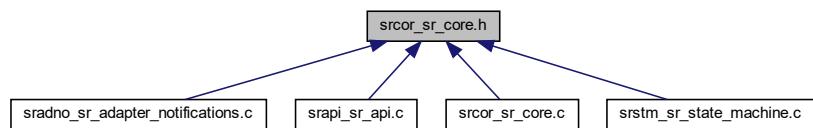
6.18 srcor_sr_core.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
#include "srtyt_sr_types.h"
```

Include dependency graph for srcor_sr_core.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct `srcor_InputBuffer`
Struct for the newly received message input buffer.
- struct `srcor_TemporaryBuffer`
Struct for the message payload temporary buffer.
- struct `srcor_RaStaConnectionData`
Struct for the process data of a RaSTA connection.

Functions

- bool `srcor_IsConfigurationValid` (const `srcty_SafetyRetransmissionConfiguration` *const `sr_layer_configuration`)
Checks if a SafRetL configuration is valid.
- void `srcor_Init` (const `srcty_SafetyRetransmissionConfiguration` *const `sr_layer_configuration`)
Initialize SafRetL core module.
- `radef_RaStaReturnCode srcor_GetConnectionId` (const `uint32_t` `sender_id`, const `uint32_t` `receiver_id`, `uint32_t` *const `connection_id`)
Get the connection identification associated with the specified sender and receiver identification.
- void `srcor_InitRaStaConnData` (const `uint32_t` `connection_id`)
Initialize the SafRetL core data of a dedicated RaSTA connection.
- void `srcor_CloseRedundancyChannel` (const `uint32_t` `connection_id`)
Close the redundancy channel of a rasta connection and send a diagnostic notification.
- void `srcor_ReceiveMessage` (const `uint32_t` `connection_id`, `srtyp_ConnectionEvents` *const `connection_event`, `bool` *const `sequence_number_in_seq`, `bool` *const `confirmed_time_stamp_in_seq`)
Read and analyze a received SafRetL message.
- bool `srcor_ProcessReceivedMessage` (const `uint32_t` `connection_id`)
Process a successfully received message of a dedicated connection.
- void `srcor_UpdateConfirmedTxSequenceNumber` (const `uint32_t` `connection_id`)
Update confirmed sequence number to transmit from the message in the input buffer.
- void `srcor_UpdateConfirmedRxSequenceNumber` (const `uint32_t` `connection_id`)
Update confirmed sequence number receive from message in input buffer.
- bool `srcor_IsProtocolVersionAccepted` (const `uint32_t` `connection_id`)
Checks if the requested protocol version is accepted.
- void `srcor_SetReceivedMessagePendingFlag` (const `uint32_t` `connection_id`)
Set the message pending flag of a dedicated RaSTA connection.
- bool `srcor_GetReceivedMessagePendingFlag` (const `uint32_t` `connection_id`)
Get the received message pending flag for a dedicated RaSTA connection.
- void `srcor_WriteMessagePayloadToTemporaryBuffer` (const `uint32_t` `connection_id`, const `uint16_t` `message_payload_size`, const `uint8_t` *const `message_payload`)
Write message payload to temporary buffer for messages to send.
- void `srcor_ClearInputBufferMessagePendingFlag` (const `uint32_t` `connection_id`)
Clear input buffer messages pending flag.
- void `srcor_SendDataMessage` (const `uint32_t` `connection_id`)
Create and send a SafRetL data message from the temporary buffer.
- void `srcor_SendConnReqMessage` (const `uint32_t` `connection_id`)
Create and send a SafRetL connection request message.
- void `srcor_SendConnRespMessage` (const `uint32_t` `connection_id`)
Create and send a SafRetL connection response message.
- void `srcor_SendDiscReqMessage` (const `uint32_t` `connection_id`, const `srtyp_DiscReason` `disconnect_reason`)

- Create and send a *SafRetL* disconnection request message.
 - void **srcor_SetDiscDetailedReason** (const uint32_t connection_id, const uint16_t detailed_disconnect_reason)
 - Set detailed reason for disconnection request message.
 - void **srcor_SendHbMessage** (const uint32_t connection_id)
 - Create and send a *SafRetL* heartbeat message.
 - void **srcor_SendRetrReqMessage** (const uint32_t connection_id)
 - Create and send a *SafRetL* retransmission request message.
 - void **srcor_HandleRetrReq** (const uint32_t connection_id)
 - Handle a retransmission request and send a *SafRetL* retransmission response, heartbeat or data message.
 - bool **srcor_IsRetrReqSequenceNumberAvailable** (const uint32_t connection_id)
 - Checks if a requested retransmission sequence number is available in the send buffer.
 - bool **srcor_IsConnRoleServer** (const uint32_t connection_id)
 - Returns true, if the own connection role is server.
 - bool **srcor_IsMessageTimeout** (const uint32_t connection_id)
 - Checks if a message timeout occurred.
 - bool **srcor_IsHeartbeatInterval** (const uint32_t connection_id)
 - Checks if the heartbeat interval is elapsed.
 - bool **srcor_IsReceivedMsgPendingAndBuffersNotFull** (const uint32_t connection_id)
 - Check if received messages are pending and send & received buffer are not full.
 - void **srcor_SendPendingMessages** (const uint32_t connection_id)
 - Send pending messages from the send buffer, if the flow control allows to send.
 - void **srcor_SendConnectionStateNotification** (const uint32_t connection_id, const **srtyp_ConnectionStates** connection_state, const **srtyp_DiscReason** disconnect_reason)
 - Send a connection state changed notification to the application layer.
 - void **srcor_GetBufferSizeAndUtilisation** (const uint32_t connection_id, **srtyp_BufferUtilisation** *const buffer_utilisation, uint16_t *const opposite_buffer_size)
 - Get the opposite receive buffer size and the own buffer utilisation for the connection state request.

Variables

- const **srtyp_ProtocolVersion** **srcor_kProtocolVersion**
 - Definition of RaSTA protocol version.

6.18.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

420e705e9ad1b43bfa1c0be45f837e0fe2573bd4

Changelog

-- Initial version (-, -)

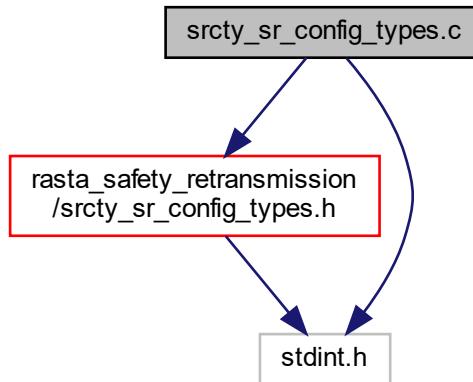
SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4933: Duplicated sent of diagnostic notification in sr_core when closing the connection (06.12.2022, N. Andres)

6.19 srcty_sr_config_types.c File Reference

Definition of RaSTA SafRetL configuration min./max. range constants.

```
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
#include <stdint.h>
```

Include dependency graph for srcty_sr_config_types.c:



Variables

- const uint32_t **srcty_kMinNumberOfRaStaConnections** = 1U
Minimum number of RaSTA connections per RaSTA network.
- const uint16_t **srcty_kMinSrLayerPayloadDataSize** = 1U
Minimum SafRetL payload data size.
- const uint32_t **srcty_kMinTMax** = 750U
Minimum max. accepted age of a message (Tmax) [ms].
- const uint32_t **srcty_kMaxTMax** = 2000U
Maximum max. accepted age of a message (Tmax) [ms].
- const uint32_t **srcty_kMinTHearbeat** = 300U
Minimum heartbeat period (Th) [ms].
- const uint32_t **srcty_kMaxTHearbeat** = 750U
Maximum heartbeat period (Th) [ms].
- const uint16_t **srcty_kMinNSendMax** = 2U
Minimum receive buffer size (Nsendmax) [messages].
- const uint16_t **srcty_kMinMWA** = 1U
Minimum max. number of received, unconfirmed messages (MWA) [messages].
- const uint16_t **srcty_kMaxMWA** = 19U
Maximum max. number of received, unconfirmed messages (MWA) [messages].
- const uint32_t **srcty_kNMaxPacket** = 1U
Packetization factor (must always be 1!).
- const uint32_t **srcty_kMinNDiagWindow** = 100U
Minimum SafRetL diagnosis window size (Ndiagwindow) [messages].
- const uint32_t **srcty_kMaxNDiagWindow** = 10000U

- const uint16_t **srcty_kByteCountUInt16** = 2U
Byte count of type UInt16_t [bytes].
- const uint16_t **srcty_kByteCountUInt32** = 4U
Byte count of type UInt32_t [bytes].
- const uint16_t **srcty_kByteCountUInt64** = 8U
Byte count of type UInt64_t [bytes].
- const uint8_t **srcty_kProtocolVersionMinValue** = 0x30
Minimum ASCII character value for protocol verion.
- const uint8_t **srcty_kProtocolVersionMaxValue** = 0x39
Maximum ASCII character value for protocol verion.
- const uint32_t **srcty_kMinFreeEntriesSendBufferForRetr** = 3U
Minimum amount of free entries in the send buffer in case of a retransmission.
- const uint32_t **srcty_kMinFreeEntriesReceivedBufferForReceive** = 1U
Minimum amount of free entries in the received buffer in case of receiving a message.

6.19.1 Detailed Description

Definition of RaSTA SafRetL configuration min./max. range constants.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

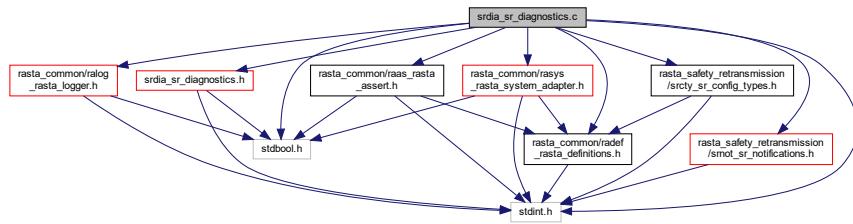
Changelog :: Initial version (-, -)

6.20 srdia_sr_diagnostics.c File Reference

Implementation of RaSTA SafRetL diagnostics module.

```
#include "srdia_sr_diagnostics.h"
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
```

```
#include "rasta_safety_retransmission/srnot_sr_notifications.h"
Include dependency graph for srdia_sr_diagnostics.c:
```



Functions

- static void `AddTimeToTimingDistribution` (uint32_t *const distribution_array, const uint32_t time_value)

Increment the matching distribution interval for a new time in a specific distribution array.
- bool `srdia_AreDiagnosticTimingIntervalsValid` (const uint32_t t_max, const uint32_t *const diag_timing_distr_intervals)

Checks if the diagnostic timings distribution intervals are valid.
- void `srdia_Init` (const uint32_t configured_connections, const uint32_t t_max, const uint32_t n_diag_window, const uint32_t *const diag_timing_distr_intervals)

Initialize SafRett diagnostics module.
- void `srdia_InitConnectionDiagnostics` (const uint32_t connection_id)

Initialize the diagnostic data of a dedicated RaSTA connection.
- void `srdia_IncSafetyCodeErrorCounter` (const uint32_t connection_id)

Increment the safety code error counter of a dedicated RaSTA connection.
- void `srdia_IncAddressErrorCounter` (const uint32_t connection_id)

Increment the address error counter of a dedicated RaSTA connection.
- void `srdia_IncTypeErrorCounter` (const uint32_t connection_id)

Increment the message type error counter of a dedicated RaSTA connection.
- void `srdia_IncSequenceNumberErrorCounter` (const uint32_t connection_id)

Increment the sequence number error counter of a dedicated RaSTA connection.
- void `srdia_IncConfirmedSequenceNumberErrorCounter` (const uint32_t connection_id)

Increment the confirmed sequence number error counter of a dedicated RaSTA connection.
- void `srdia_UpdateConnectionDiagnostics` (const uint32_t connection_id, const uint32_t round_trip_delay, const uint32_t alive_time)

Update the received message timing statistics of a dedicated RaSTA connection.
- void `srdia_SendDiagnosticNotification` (const uint32_t connection_id)

Send a diagnostic notification of a dedicated RaSTA connection.

Variables

- **PRIVATE** bool `srdia_initialized` = false

Initialization state of the module. true, if the module is initialized.
- **PRIVATE** uint32_t `srdia_number_of_connections` = 0U

Number of configured RaSTA connections.
- **PRIVATE** uint32_t `srdia_t_max` = 0U

Maximal accepted age of a message [ms].
- **PRIVATE** uint32_t `srdia_n_diag_window` = 0U

Configured diagnosis window size [messages].

- **PRIVATE** uint32_t **srdia_diag_timing_distr_intervals** [**RADEF_DIAGNOSTIC_TIMING_DISTRIBUTION_ARRAY_SIZE**]
Diagnostic timing distribution intervals [ms].
- **PRIVATE** srdia_SrConnectionDiagnostics **srdia_connection_diagnostics** [**RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS**]
Received buffers for all RaSTA connections.
- **PRIVATE** uint16_t **srdia_logger_id**
ID of the sr_diagnostics debug logger.

6.20.1 Detailed Description

Implementation of RaSTA SafRetL diagnostics module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

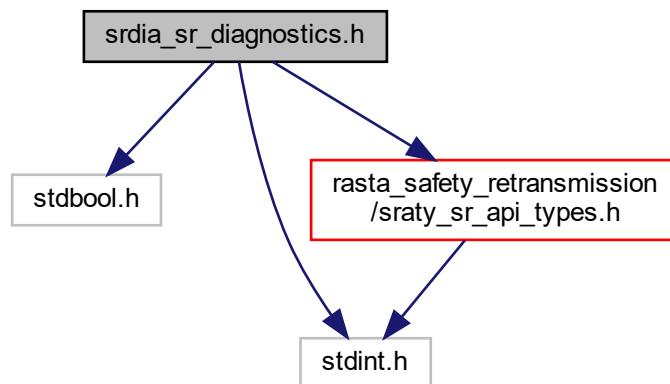
e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog

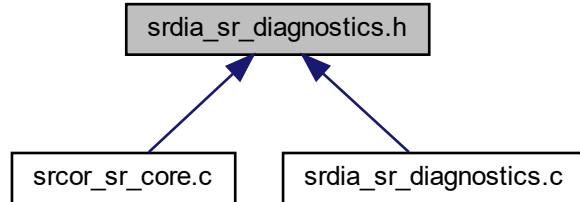
:: Initial version (-, -)

6.21 srdia_sr_diagnostics.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
Include dependency graph for srdia_sr_diagnostics.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct [srdia_SrConnectionDiagnostics](#)
Struct for the collection of diagnostic data of a RaSTA connection.

Functions

- bool [srdia_AreDiagnosticTimingIntervalsValid](#) (const uint32_t t_max, const uint32_t *const diag_timing_distr_intervals)
Checks if the diagnostic timings distribution intervals are valid.
- void [srdia_Init](#) (const uint32_t configured_connections, const uint32_t t_max, const uint32_t n_diag_window, const uint32_t *const diag_timing_distr_intervals)
Initialize SafRetL diagnostics module.
- void [srdia_InitConnectionDiagnostics](#) (const uint32_t connection_id)
Initialize the diagnostic data of a dedicated RaSTA connection.
- void [srdia_IncSafetyCodeErrorCounter](#) (const uint32_t connection_id)
Increment the safety code error counter of a dedicated RaSTA connection.
- void [srdia_IncAddressErrorCounter](#) (const uint32_t connection_id)
Increment the address error counter of a dedicated RaSTA connection.
- void [srdia_IncTypeErrorHandler](#) (const uint32_t connection_id)
Increment the message type error counter of a dedicated RaSTA connection.
- void [srdia_IncSequenceNumberErrorCounter](#) (const uint32_t connection_id)
Increment the sequence number error counter of a dedicated RaSTA connection.
- void [srdia_IncConfirmedSequenceNumberErrorCounter](#) (const uint32_t connection_id)
Increment the confirmed sequence number error counter of a dedicated RaSTA connection.
- void [srdia_UpdateConnectionDiagnostics](#) (const uint32_t connection_id, const uint32_t round_trip_delay, const uint32_t alive_time)
Update the received message timing statistics of a dedicated RaSTA connection.
- void [srdia_SendDiagnosticNotification](#) (const uint32_t connection_id)
Send a diagnostic notification of a dedicated RaSTA connection.

6.21.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

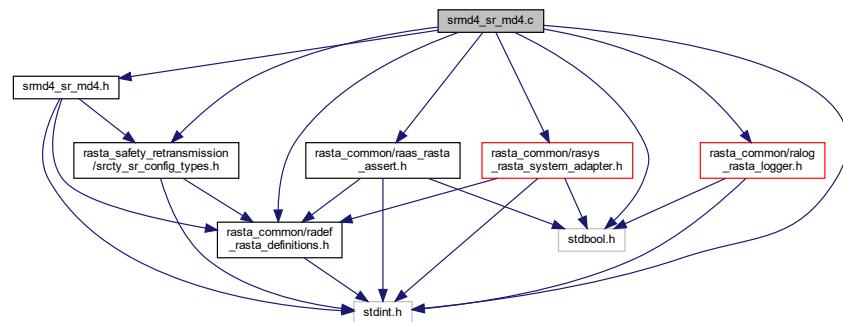
e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog :: Initial version (-, -)

6.22 srmd4_sr_md4.c File Reference

Implementation of RaSTA SafRetL MD4 module.

```
#include "srmd4_sr_md4.h"
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
Include dependency graph for srmd4_sr_md4.c:
```



Classes

- struct [Md4Context](#)
Structure holding the context data for a MD4 calculation.

Macros

- `#define MD4_INPUT_DATA_BLOCK_SIZE (64U)`
Size of a input data block used for the MD4 calculation [bytes].
- `#define BYTES_PER_U32 (4U)`
Bytes per uint32_t [bytes].

Enumerations

- enum `Md4Function` { `kMd4FunctionF` = 0U , `kMd4FunctionG` = 1U , `kMd4FunctionH` = 2U }
- Enum for the selection of the basic MD4 functions.*

Functions

- static void `Md4Body` (`Md4Context` *const ctx, const `uint8_t` *const data, const `uint32_t` number_of_blocks)
Inner function for MD4 calculation.
- static void `Md4Update` (`Md4Context` *const ctx, const `uint8_t` *const data, const `uint32_t` size)
Calculation of MD4. Allows incremental calculation.
- static void `Md4Final` (`Md4Context` *ctx, `uint8_t` *result)
Final MD4 calculations, generating the MD4 hash.
- static void `SetContextBuffer` (const `uint8_t` value, const `uint32_t` size, const `uint32_t` md4_context_buffer_start_index, `Md4Context` *const md4_context)
Helper function to set the MD4 context data buffer memory to a dedicated value.
- static void `CopyToContextBuffer` (const `uint8_t` *const source, const `uint32_t` size, const `uint32_t` md4_context_buffer_start_index, `Md4Context` *const md4_context)
Helper function to copy memory to a MD4 context data buffer memory.
- static void `ClearMd4ContextData` (`Md4Context` *const md4_context)
Helper function to clear the MD4 context data.
- static void `WriteU32toByteArray` (const `uint32_t` source, `uint8_t` *const destination)
Helper to write a Uint32 value in a byte array.
- static `uint32_t` `FunctionF` (const `uint32_t` x, const `uint32_t` y, const `uint32_t` z)
Basic MD4 calculation function F, which returns $F = z \wedge (x \& (y \wedge z))$.
- static `uint32_t` `FunctionG` (const `uint32_t` x, const `uint32_t` y, const `uint32_t` z)
Basic MD4 calculation function G, which returns $G = (x \& (y \mid z)) \mid (y \& z)$.
- static `uint32_t` `FunctionH` (const `uint32_t` x, const `uint32_t` y, const `uint32_t` z)
Basic MD4 calculation function H, which returns $H = x \wedge y \wedge z$.
- static void `Step` (const `Md4Function` md4_function, `uint32_t` *const a, const `uint32_t` b, const `uint32_t` c, const `uint32_t` d, const `uint32_t` x, const `uint32_t` s)
The MD4 transformation function for all three rounds.
- static `uint32_t` `Set` (const `uint8_t` index_32_bit, const `uint8_t` *const data, `Md4Context` *const ctx)
Reads 4 input data bytes and stores them in a properly aligned Uint32 value in a MD4 context data block and returns this Uint32 value.
- static `uint32_t` `Get` (const `uint8_t` index_32_bit, const `Md4Context` *const ctx)
Get a properly aligned Uint32 value from a MD4 context data block.
- void `srmd4_CalculateMd4` (const `srcty_Md4InitValue` md4_initial_value, const `uint16_t` data_size, const `uint8_t` *const data_buffer, `srmd4_Md4` *const calculated_md4)
Calculate the MD4 of a data buffer.

Variables

- **PRIVATE Md4Context srmd4_context**
Context of MD4 calculation.
- static const uint32_t **kDataBlockSizeBits** = 6U
Number of bits for the size of a data block used for the MD4 calculation [bits].
- static const uint32_t **kDataBlockSizeBitMask** = 0x3FU
Bits used for masking the size of a data block used for the MD4 calculation.
- static const uint32_t **kDataBitCountLowIndex** = 56U
Data block index of lower data bit counter [bytes].
- static const uint32_t **kDataBitCountHighIndex** = 60U
Data block index of upper data bit counter [bytes].
- static const uint32_t **kAc1** = 0x5A827999U
Constant for [FunctionG](#) calculation.
- static const uint32_t **kAc2** = 0x6ED9EBA1U
Constant for [FunctionH](#) calculation.

6.22.1 Detailed Description

Implementation of RaSTA SafRetL MD4 module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

[Changelog](#) :: Initial version (-, -)

6.22.2 Enumeration Type Documentation

6.22.2.1 Md4Function enum [Md4Function](#)

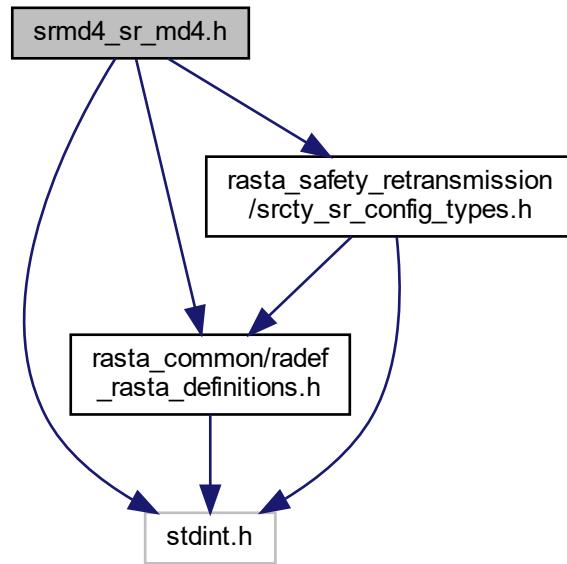
Enum for the selection of the basic MD4 functions.

Enumerator

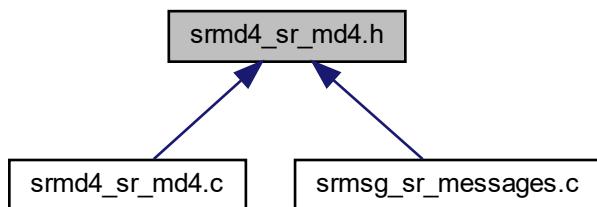
kMd4FunctionF	Basic MD4 function F.
kMd4FunctionG	Basic MD4 function G.
kMd4FunctionH	Basic MD4 function H.

6.23 srmd4_sr_md4.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
Include dependency graph for srmd4_sr_md4.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `srmd4_Md4`

Typedef for MD4 result.

Functions

- void `srmd4_CalculateMd4` (const `srcty_Md4InitValue` `md4_initial_value`, const `uint16_t` `data_size`, const `uint8_t` *const `data_buffer`, `srmd4_Md4` *const `calculated_md4`)
Calculate the MD4 of a data buffer.

6.23.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog

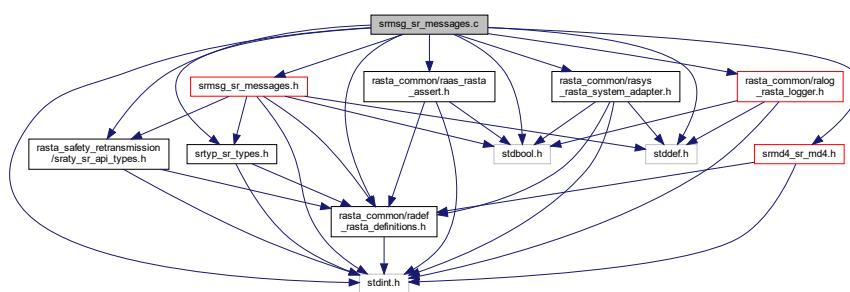
:: Initial version (-, -)

6.24 srmsg_sr_messages.c File Reference

Implementation of RaSTA SafRetL messages module.

```
#include "srmsg_sr_messages.h"
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
#include "srmd4_sr_md4.h"
#include "srtyp_sr_types.h"
```

Include dependency graph for srmsg_sr_messages.c:



Functions

- static bool `IsMessageTypeValid (srty_SrMessageType message_type)`
Checks if a message type is valid or not.
- static void `SetUint16InMessage (const uint16_t position, const uint16_t data, srty_SrMessage *const sr_message)`
Set a Uint16 at a specific position in a message.
- static void `SetUint32InMessage (const uint16_t position, const uint32_t data, srty_SrMessage *const sr_message)`
Set a Uint32 at a specific position in a message.
- static void `SetUint64InMessage (const uint16_t position, const uint64_t data, srty_SrMessage *const sr_message)`
Set a Uint64 at a specific position in a message.
- static void `SetMessageHeaderInMessage (const uint16_t message_length, const uint16_t message_type, const srty_SrMessageHeaderCreate message_header, srty_SrMessage *const sr_message)`
Set the message header data in a message.
- static void `SetProtocolVersionInMessage (const srty_ProtocolVersion protocol_version, srty_SrMessage *const sr_message)`
Set the protocol version in a message.
- static void `SetPayloadDataInMessage (const uint16_t position, const srty_SrMessagePayload *const message_payload, srty_SrMessage *const sr_message)`
Set the payload data in a message.
- static uint16_t `GetUint16FromMessage (const srty_SrMessage *const sr_message, const uint16_t position)`
Get a Uint16 from a specific position in a message.
- static uint32_t `GetUint32FromMessage (const srty_SrMessage *const sr_message, const uint16_t position)`
Get a Uint32 from a specific position in a message.
- static uint16_t `GetSafetyCodeLength (void)`
Get the length of the configured safety code.
- void `srmsg_Init (const srcty_SafetyCodeType configured_safety_code_type, const srcty_Md4InitValue configured_md4_initial_value)`
Initialize SafRetL messages module.
- void `srmsg_CreateConnReqMessage (const srty_SrMessageHeaderCreate message_header, const srty_ProtocolVersion protocol_version, const uint16_t n_send_max, srty_SrMessage *const sr_message)`
Create a new SafRetL connection request message.
- void `srmsg_CreateConnRespMessage (const srty_SrMessageHeaderCreate message_header, const srty_ProtocolVersion protocol_version, const uint16_t n_send_max, srty_SrMessage *const sr_message)`
Create a new SafRetL connection response message.
- void `srmsg_CreateDataMessage (const srty_SrMessageHeaderCreate message_header, const srty_SrMessagePayload *const message_payload, srty_SrMessage *const sr_message)`
Create a new SafRetL data message.
- void `srmsg_CreateRetrDataMessage (const srty_SrMessageHeaderCreate message_header, const srty_SrMessagePayload *const message_payload, srty_SrMessage *const sr_message)`
Create a new SafRetL retransmitted data message.
- void `srmsg_CreateRetrReqMessage (const srty_SrMessageHeaderCreate message_header, srty_SrMessage *const sr_message)`
Create a new SafRetL retransmission request message.
- void `srmsg_CreateRetrRespMessage (const srty_SrMessageHeaderCreate message_header, srty_SrMessage *const sr_message)`
Create a new SafRetL retransmission response message.
- void `srmsg_CreateHeartbeatMessage (const srty_SrMessageHeaderCreate message_header, srty_SrMessage *const sr_message)`
Create a new SafRetL heartbeat message.

- void `srmsg_CreateDiscReqMessage` (const `srtyp_SrMessageHeaderCreate` message_header, const `uint16_t` detailed_reason, const `srtyp_DiscReason` reason, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL disconnection request message.
- void `srmsg_UpdateMessageHeader` (const `srtyp_SrMessageHeaderUpdate` message_header_update, `srtyp_SrMessage` *const sr_message)

Update a SafRetL message header and calculate the safety code to prepare the message for sending.
- `radef_RaStaReturnCode srmsg_CheckMessage` (const `srtyp_SrMessage` *const sr_message)

Check MD4, message type and message size of a SafRetL PDU message.
- void `srmsg_GetMessageHeader` (const `srtyp_SrMessage` *const sr_message, `srtyp_SrMessageHeader` *const message_header)

Get the header of a SafRetL PDU message.
- `srtyp_SrMessageType srmsg_GetMessageType` (const `srtyp_SrMessage` *const sr_message)

Get the message type of a SafRetL PDU message.
- `uint32_t srmsg_GetMessageSequenceNumber` (const `srtyp_SrMessage` *const sr_message)

Get the sequence number of a SafRetL PDU message.
- void `srmsg_GetDataMessagePayload` (const `srtyp_SrMessage` *const sr_message, `srtyp_SrMessagePayload` *const message_payload)

Get the payload of a SafRetL data or retransmitted data message.
- void `srmsg_GetConnMessageData` (const `srtyp_SrMessage` *const sr_message, `srtyp_ProtocolVersion` *const protocol_version, `uint16_t` *const n_send_max)

Get the data of a SafRetL connection request or connection response message.
- void `srmsg_GetDiscMessageData` (const `srtyp_SrMessage` *const sr_message, `uint16_t` *const detailed_reason, `srtyp_DiscReason` *const reason)

Get the data of a SafRetL disconnection request message.

Variables

- **PRIVATE** bool `srmsg_initialized` = false

Initialization state of the module. true, if the module is initialized.
- **PRIVATE** `srcty_SafetyCodeType` `srmsg_safety_code_type`

Configured safety code type.
- **PRIVATE** `srcty_Md4InitValue` `srmsg_md4_initial_value`

Configured MD4 initial code value.
- static const `uint16_t` `kSafetyCodeNoneLength` = 0U

Length of no safety code [bytes].
- static const `uint16_t` `kSafetyCodeLowerMd4Length` = 8U

Length of lower MD4 safety code [bytes].
- static const `uint16_t` `kSafetyCodeFullMd4Length` = 16U

Length of full MD4 safety code [bytes].
- static const `uint16_t` `kMinMsgLengthConnReqResp` = 42U

Minimum message length for connection request & response message [byte].
- static const `uint16_t` `kMinMsgLengthRetrReqResp` = 28U

Minimum message length for retransmission request & response message [byte].
- static const `uint16_t` `kMinMsgLengthDiscReq` = 32U

Minimum message length for disconnection request message [byte].
- static const `uint16_t` `kMinMsgLengthHeartbeat` = 28U

Minimum message length for heartbeat message [byte].
- static const `uint16_t` `kMinMsgLengthEmptyDataMsg` = 30U

Minimum message length for an empty data or retransmitted data message [byte].
- static const `uint16_t` `kMsgLengthPosition` = 0U

Start position for message length in PDU message.

- static const uint16_t **kMsgTypePosition** = 2U
Start position for message type in PDU message.
- static const uint16_t **kMsgReceiverPosition** = 4U
Start position for receiver identification in PDU message.
- static const uint16_t **kMsgSenderPosition** = 8U
Start position for sender identification in PDU message.
- static const uint16_t **kMsgSequenceNbrPosition** = 12U
Start position for sequence number in PDU message.
- static const uint16_t **kMsgConfirmedSequenceNbrPosition** = 16U
Start position for confirmed sequence number in PDU message.
- static const uint16_t **kMsgTimeStampPosition** = 20U
Start position for time stamp in PDU message.
- static const uint16_t **kMsgConfirmedTimeStampPosition** = 24U
Start position for confirmed time stamp in PDU message.
- static const uint16_t **kMsgProtocolVersionByteSize** = 4U
Size of the protocol version [byte].
- static const uint16_t **kMsgProtocolVersionPosition** = 28U
Start position for protocol version in connection PDU message.
- static const uint16_t **kMsgNsendmaxPosition** = 32U
Start position for Nsendmax in connection PDU message.
- static const uint16_t **kMsgReserveParameterPosition** = 34U
Start position for reserve parameter in connection PDU message.
- static const uint16_t **kMsgPayloadDataSizePosition** = 28U
Start position for payload data size in data PDU message.
- static const uint16_t **kMsgPayloadDataPosition** = 30U
Start position for payload data in data PDU message.
- static const uint16_t **kMsgDetailedInfosDisconnectPosition**
Start position for detailed informations for disconnection reason in disconnection PDU message.
- static const uint16_t **kMsgReasonDisconnectPosition** = 30U
Start position for reason for disconnection request in disconnection PDU message.

6.24.1 Detailed Description

Implementation of RaSTA SafRetL messages module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

4278c3bc2a5a128fd5eeaaa2c5d939b42ea82149

Changelog

-- Initial version (-, -)

SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4935: Messages module must check message size and payload data size correctly (07.12.2022, N. Andres)

6.24.2 Variable Documentation

6.24.2.1 kMsgDetailedInfosDisconnectPosition const uint16_t kMsgDetailedInfosDisconnectPosition
[static]

Initial value:

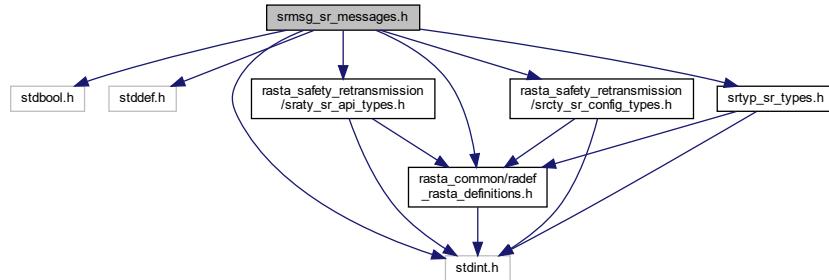
= 28U

Start position for detailed informations for disconnection reason in disconnection PDU message.

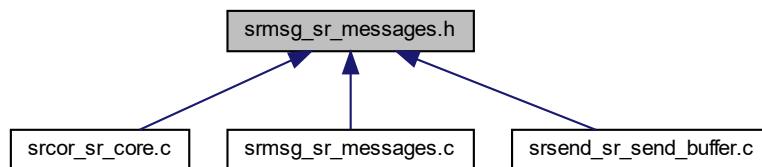
6.25 srmsg_sr_messages.h File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_safety_retransmission/srty_sr_api_types.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
#include "srtyt_sr_types.h"
```

Include dependency graph for srmsg_sr_messages.h:



This graph shows which files directly or indirectly include this file:



Functions

- void `srmsg_Init` (const `srcty_SafetyCodeType` configured_safety_code_type, const `srcty_Md4InitValue` configured_md4_initial_value)

Initialize SafRetL messages module.
- void `srmsg_CreateConnReqMessage` (const `srtyp_SrMessageHeaderCreate` message_header, const `srtyp_ProtocolVersion` protocol_version, const `uint16_t` n_send_max, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL connection request message.
- void `srmsg_CreateConnRespMessage` (const `srtyp_SrMessageHeaderCreate` message_header, const `srtyp_ProtocolVersion` protocol_version, const `uint16_t` n_send_max, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL connection response message.
- void `srmsg_CreateDataMessage` (const `srtyp_SrMessageHeaderCreate` message_header, const `srtyp_SrMessagePayload` *const message_payload, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL data message.
- void `srmsg_CreateRetrDataMessage` (const `srtyp_SrMessageHeaderCreate` message_header, const `srtyp_SrMessagePayload` *const message_payload, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL retransmitted data message.
- void `srmsg_CreateRetrReqMessage` (const `srtyp_SrMessageHeaderCreate` message_header, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL retransmission request message.
- void `srmsg_CreateRetrRespMessage` (const `srtyp_SrMessageHeaderCreate` message_header, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL retransmission response message.
- void `srmsg_CreateHeartbeatMessage` (const `srtyp_SrMessageHeaderCreate` message_header, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL heartbeat message.
- void `srmsg_CreateDiscReqMessage` (const `srtyp_SrMessageHeaderCreate` message_header, const `uint16_t` detailed_reason, const `srtyp_DiscReason` reason, `srtyp_SrMessage` *const sr_message)

Create a new SafRetL disconnection request message.
- void `srmsg_UpdateMessageHeader` (const `srtyp_SrMessageHeaderUpdate` message_header_update, `srtyp_SrMessage` *const sr_message)

Update a SafRetL message header and calculate the safety code to prepare the message for sending.
- `radef_RaStaReturnCode srmsg_CheckMessage` (const `srtyp_SrMessage` *const sr_message)

Check MD4, message type and message size of a SafRetL PDU message.
- `srtyp_SrMessageType srmsg_GetMessageType` (const `srtyp_SrMessage` *const sr_message)

Get the message type of a SafRetL PDU message.
- `uint32_t srmsg_GetMessageSequenceNumber` (const `srtyp_SrMessage` *const sr_message)

Get the sequence number of a SafRetL PDU message.
- void `srmsg_GetDataMessagePayload` (const `srtyp_SrMessage` *const sr_message, `srtyp_SrMessagePayload` *const message_payload)

Get the payload of a SafRetL data or retransmitted data message.
- void `srmsg_GetConnMessageData` (const `srtyp_SrMessage` *const sr_message, `srtyp_ProtocolVersion` *const protocol_version, `uint16_t` *const n_send_max)

Get the data of a SafRetL connection request or connection response message.
- void `srmsg_GetDiscMessageData` (const `srtyp_SrMessage` *const sr_message, `uint16_t` *const detailed_reason, `srtyp_DiscReason` *const reason)

Get the data of a SafRetL disconnection request message.

6.25.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

87bc446bbb57fb82711f48fb5ccb62bccae929e8

Changelog

--: Initial version (-, -)

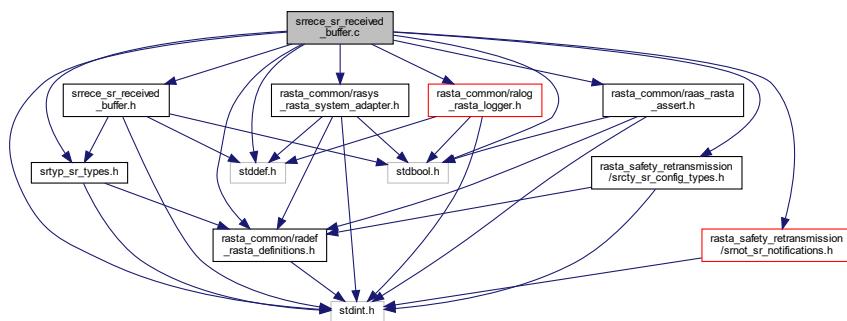
SBB-RaSTA-083-SoftwareChangeRecord-001: BUG 4935: Messages module must check message size and payload data size correctly (07.12.2022, N. Andres)

6.26 srrece_sr_received_buffer.c File Reference

Implementation of RaSTA SafRetL received buffer module.

```
#include "srrece_sr_received_buffer.h"
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
#include "rasta_safety_retransmission/srnot_sr_notifications.h"
#include "srtyp_sr_types.h"
```

Include dependency graph for srrece_sr_received_buffer.c:



Classes

- struct **SrReceivedBuffer**
Struct for SafRetL received messages payload buffer.

Functions

- static void [IncrementReceivedBufferIndexAndHandleOverflow](#) (uint16_t *const bufferIndex, const uint16_t increment)
Increment a received buffer index by a increment value and handle overflow.
- void [srrece_Init](#) (const uint32_t configured_connections, const uint16_t configured_n_send_max)
Initialize the SafRetL received buffer module.
- void [srrece_InitBuffer](#) (const uint32_t connection_id)
Initialize the received buffer of a dedicated RaSTA connection.
- void [srrece_AddToBuffer](#) (const uint32_t connection_id, const [srtyp_SrMessagePayload](#) *const message_payload)
Add a SafRetL message to the received buffer of a dedicated RaSTA connection. A fatal error is raised, if the buffer is full.
- [radef_RaStaReturnCode srrece_ReadFromBuffer](#) (const uint32_t connection_id, [srtyp_SrMessagePayload](#) *const message_payload)
Read and remove a SafRetL message payload from the received buffer of a dedicated RaSTA connection.
- uint32_t [srrece_GetPayloadSizeOfNextMessageToRead](#) (const uint32_t connection_id)
Get the payload size of the next message that is read from a dedicated RaSTA connection.
- uint16_t [srrece_GetFreeBufferEntries](#) (const uint32_t connection_id)
Get the number of free buffer entries.
- uint16_t [srrece_GetUsedBufferEntries](#) (const uint32_t connection_id)
Get the number of used buffer entries.

Variables

- **PRIVATE** bool [srrece_initialized](#) = false
Initialization state of the module. true, if the module is initialized.
- **PRIVATE** uint32_t [srrece_number_of_connections](#) = 0U
Number of configured RaSTA connections.
- **PRIVATE** uint16_t [srrece_n_send_max](#) = 0U
Configured receive buffer size [messages].
- **PRIVATE** SrReceivedBuffer [srrece_received_buffers](#) [[RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS](#)]
Received buffers for all RaSTA connections.
- static const uint16_t [kIndexIncrement](#) = 1U
Increment value for received buffer indexes.

6.26.1 Detailed Description

Implementation of RaSTA SafRetL received buffer module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

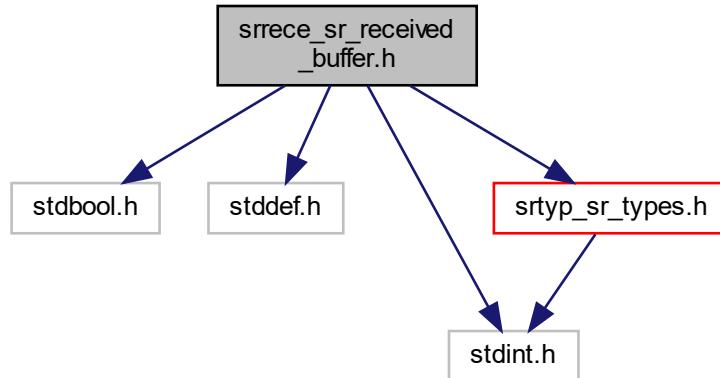
e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog

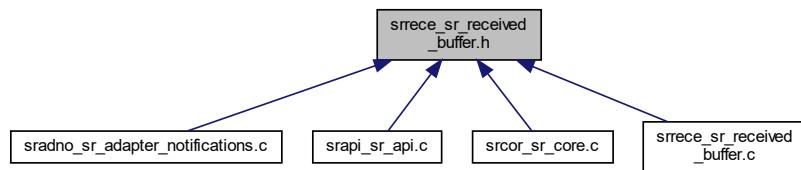
--: Initial version (-, -)

6.27 srrece_sr_received_buffer.h File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "srtyp_sr_types.h"
Include dependency graph for srrece_sr_received_buffer.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- `void srrece_Init (const uint32_t configured_connections, const uint16_t configured_n_send_max)`
Initialize the SafRetL received buffer module.
- `void srrece_InitBuffer (const uint32_t connection_id)`
Initialize the received buffer of a dedicated RaSTA connection.
- `void srrece_AddToBuffer (const uint32_t connection_id, const srtyp_SrMessagePayload *const message_payload)`
Add a SafRetL message to the received buffer of a dedicated RaSTA connection. A fatal error is raised, if the buffer is full.
- `radef_RaStaReturnCode srrece_ReadFromBuffer (const uint32_t connection_id, srtyp_SrMessagePayload *const message_payload)`
Read and remove a SafRetL message payload from the received buffer of a dedicated RaSTA connection.

- `uint32_t srrece_GetPayloadSizeOfNextMessageToRead (const uint32_t connection_id)`
Get the payload size of the next message that is read from a dedicated RaSTA connection.
- `uint16_t srrece_GetFreeBufferEntries (const uint32_t connection_id)`
Get the number of free buffer entries.
- `uint16_t srrece_GetUsedBufferEntries (const uint32_t connection_id)`
Get the number of used buffer entries.

6.27.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

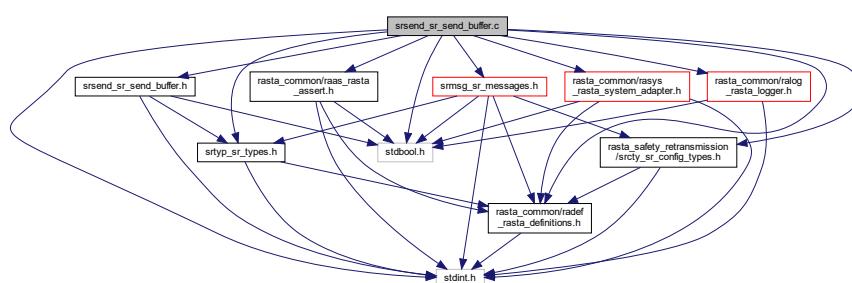
Changelog

-- Initial version (-, -)

6.28 srsend_sr_send_buffer.c File Reference

Implementation of RaSTA SafRetL send buffer module.

```
#include "srsend_sr_send_buffer.h"
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
#include "srmsg_sr_messages.h"
#include "srtyp_sr_types.h"
Include dependency graph for srsend_sr_send_buffer.c:
```



Classes

- struct [SrSendMessage](#)
Struct for SafRetL send message.
- struct [SrSendBuffer](#)
Struct for SafRetL send messages buffer.

Functions

- static void [InitBuffer](#) (const uint32_t connection_id)
Initialize the send buffer of a dedicated RaSTA connection.
- static void [AddToBuffer](#) (const uint32_t connection_id, const [srtyp_SrMessage](#) *const message)
Add a SafRetL message to the send buffer of a dedicated RaSTA connection.
- static void [CopyTempBufferIntoConnectionBuffer](#) (const uint32_t connection_id)
Copy the temporary buffer into a connection buffer.
- static void [CreateNewDataMsgAndAddToTempBuffer](#) (const [srtyp_SrMessage](#) *const sr_message, [srtyp_SrMessageHeaderCreate](#) *const new_msg_header)
Create a new Data message and add it to the temporary buffer.
- static void [CreateNewRetrDataMsgAndAddToTempBuffer](#) (const [srtyp_SrMessage](#) *const sr_message, [srtyp_SrMessageHeaderCreate](#) *const new_msg_header)
Create a new RetrData message and add it to the temporary buffer.
- static void [CreateNewHbMsgAndAddToTempBuffer](#) ([srtyp_SrMessageHeaderCreate](#) *const new_msg_header)
Create a new heartbeat message and add it to the temporary buffer.
- static void [CreateNewRetrReqMsgAndAddToTempBuffer](#) ([srtyp_SrMessageHeaderCreate](#) *const new_msg_header)
Create a new RetrReq message and add it to the temporary buffer.
- static void [IncrementSendBufferIndexAndHandleOverflow](#) (uint16_t *const bufferIndex, const uint16_t increment)
Increment a send buffer index by a increment value and handle overflow.
- void [srsend_Init](#) (const uint32_t configured_connections)
Initialize all data of the SafRetL send buffer module.
- void [srsend_InitBuffer](#) (const uint32_t connection_id)
Initialize the send buffer of a dedicated RaSTA connection.
- void [srsend_AddToBuffer](#) (const uint32_t connection_id, const [srtyp_SrMessage](#) *const message)
Add a SafRetL message to the send buffer of a dedicated RaSTA connection.
- [raodef_RaStaReturnCode](#) [srsend_ReadMessageToSend](#) (const uint32_t connection_id, [srtyp_SrMessage](#) *const message)
Read a SafRetL message from the send buffer of a dedicated RaSTA connection.
- void [srsend_PreparesBufferForRetr](#) (const uint32_t connection_id, const uint32_t sequence_number_for_retransmission, const [srtyp_SrMessageHeaderCreate](#) message_header, uint32_t *const new_current_sequence_number)
Prepare send buffer for retransmission starting with a defined sequence number.
- [raodef_RaStaReturnCode](#) [srsend_IsSequenceNumberInBuffer](#) (const uint32_t connection_id, const uint32_t sequence_number)
Checks if a message with a specific sequence number is in the send buffer.
- void [srsend_RemoveFromBuffer](#) (const uint32_t connection_id, const uint32_t confirmed_sequence_number)
Remove confirmed SafRetL messages from the send buffer from a defined sequence number.
- uint16_t [srsend_GetFreeBufferEntries](#) (const uint32_t connection_id)
Get the number of free buffer entries.
- uint16_t [srsend_GetUsedBufferEntries](#) (const uint32_t connection_id)
Get the number of used buffer entries.
- uint16_t [srsend_GetNumberOfMessagesToSend](#) (const uint32_t connection_id)
Get the number of messages to send from the send buffer.

Variables

- **PRIVATE** bool **srsend_initialized** = false
Initialization state of the module. true, if the module is initialized.
- **PRIVATE** uint32_t **srsend_number_of_connections** = 0U
Number of configured RaSTA connections.
- **PRIVATE** SrSendBuffer **srsend_send_buffers** [RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS+1U]
Send buffers for all RaSTA connections.
- **PRIVATE** srtyp_SrMessage **scratch_message**
Memory to transfer and create messages. Valid range as described in [srtyp_SrMessage](#).
- **PRIVATE** srtyp_SrMessagePayload **scratch_message_payload**
Memory to transfer and create messages payload. Valid range as described in [srtyp_SrMessagePayload](#).
- static const uint16_t **kIndexIncrement** = 1U
Increment value for send buffer indexes.

6.28.1 Detailed Description

Implementation of RaSTA SafRetL send buffer module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog :: Initial version (-, -)

6.28.2 Variable Documentation

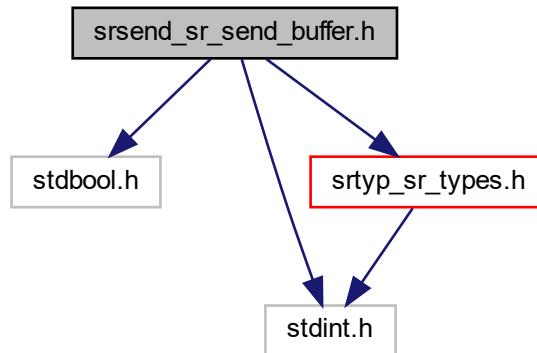
6.28.2.1 **srsend_send_buffers** **PRIVATE** SrSendBuffer **srsend_send_buffers**[RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS+1U]

Send buffers for all RaSTA connections.

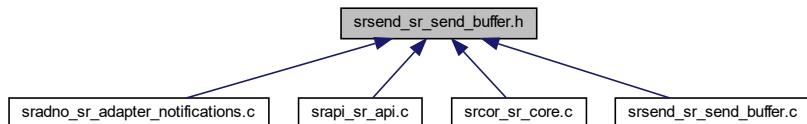
This array contains a buffer for every RaSTA connection and additionally a temporary buffer for the preparation of the send buffer for a retransmission.

6.29 srsend_sr_send_buffer.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "srtyp_sr_types.h"
Include dependency graph for srsend_sr_send_buffer.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- void `srsend_Init` (const uint32_t configured_connections)

Initialize all data of the SafRetL send buffer module.
- void `srsend_InitBuffer` (const uint32_t connection_id)

Initialize the send buffer of a dedicated RaSTA connection.
- void `srsend_AddToBuffer` (const uint32_t connection_id, const `srtyp_SrMessage` *const message)

Add a SafRetL message to the send buffer of a dedicated RaSTA connection.
- `radef_RaStaReturnCode srsend_ReadMessageToSend` (const uint32_t connection_id, `srtyp_SrMessage` *const message)

Read a SafRetL message from the send buffer of a dedicated RaSTA connection.
- void `srsend_PrepBufferForRetr` (const uint32_t connection_id, const uint32_t sequence_number_for_retransmission, const `srtyp_SrMessageHeaderCreate` message_header, uint32_t *const new_current_sequence_number)

Prepare send buffer for retransmission starting with a defined sequence number.

- `radef_RaStaReturnCode srsend_IsSequenceNumberInBuffer (const uint32_t connection_id, const uint32_t sequence_number)`
Checks if a message with a specific sequence number is in the send buffer.
- `void srsend_RemoveFromBuffer (const uint32_t connection_id, const uint32_t confirmed_sequence_number)`
Remove confirmed SafRetL messages from the send buffer from a defined sequence number.
- `uint16_t srsend_GetFreeBufferEntries (const uint32_t connection_id)`
Get the number of free buffer entries.
- `uint16_t srsend_GetUsedBufferEntries (const uint32_t connection_id)`
Get the number of used buffer entries.
- `uint16_t srsend_GetNumberOfMessagesToSend (const uint32_t connection_id)`
Get the number of messages to send from the send buffer.

6.29.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

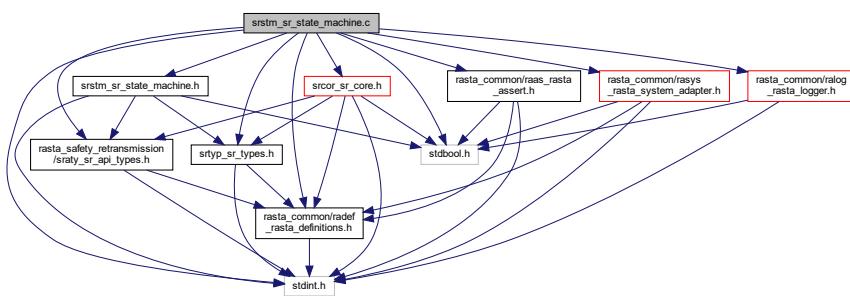
e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog :: Initial version (-, -)

6.30 srstm_sr_state_machine.c File Reference

Implementation of RaSTA SafRetL state machine module.

```
#include "srstm_sr_state_machine.h"
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
#include "srcor_sr_core.h"
#include "srtyp_sr_types.h"
Include dependency graph for srstm_sr_state_machine.c:
```



Functions

- static void `ProcessStateClosedEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event)
Process events in state Closed.
- static void `ProcessStateDownEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event)
Process events in state Down.
- static void `ProcessStateStartEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process events in state Start.
- static void `ProcessStateUpEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process events in state Up.
- static void `ProcessStateRetransRequestEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq)
Process events in state RetransRequest.
- static void `ProcessStateRetransRunningEvents` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process events in state RetransRunning.
- static bool `ProcessReceivedMessage` (const uint32_t connection_id)
Process received message and evaluate timeliness.
- static void `CloseConnection` (const uint32_t connection_id, const `srtty_DiscReason` disconnect_reason, const bool is_incoming_message)
Close the connection of a specific RaSTA connection.
- static void `CloseRedundancyChannel` (const uint32_t connection_id, const bool is_incoming_message)
Close the redundancy channel of a specific RaSTA connection.
- static void `StartRetransmission` (const uint32_t connection_id, const `srtty_ConnectionStates` new_state, bool retransmission_requested)
Start a retransmission.
- static void `UpdateConnectionState` (const uint32_t connection_id, const `srtty_ConnectionStates` new_state)
Update the connection state and send a connection state notification to the application layer.
- static void `UpdateConnectionStateWithDiscReason` (const uint32_t connection_id, const `srtty_ConnectionStates` new_state, const `srtty_DiscReason` disconnect_reason)
Update the connection state and send a connection state notification to the application layer.
- void `srstm_Init` (const uint32_t configured_connections)
Initialize SafRetL state machine module.
- void `srstm_ProcessConnectionStateMachine` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process RaSTA connection state machine.
- `srtty_ConnectionStates srstm_GetConnectionState` (const uint32_t connection_id)
Return the state of a RaSTA connection state machine.

Variables

- `PRIVATE` bool `srstm_initialized` = false
Initialization state of the module. true, if the module is initialized.
- `PRIVATE` uint32_t `srstm_number_of_connections` = 0U
Number of configured RaSTA connections.
- `PRIVATE` `srtty_ConnectionStates` `srstm_connection_states` [RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS]
State machine states of the RaSTA connections.
- `PRIVATE` `srtty_ConnectionStates` `srstm_connection_states_old` [RADEF_MAX_NUMBER_OF_RASTA_CONNECTIONS]
Old state machine states of the RaSTA connections.

6.30.1 Detailed Description

Implementation of RaSTA SafRetL state machine module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

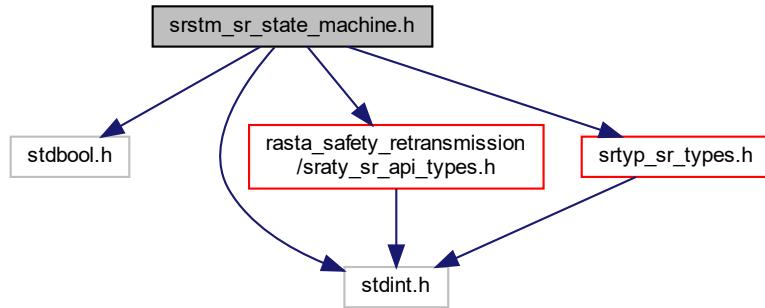
e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog :: Initial version (-, -)

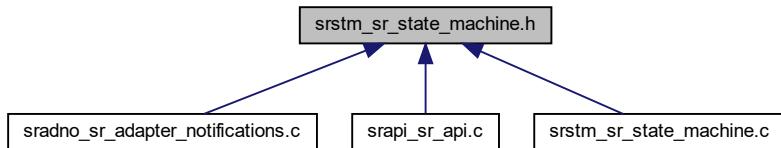
6.31 srstm_sr_state_machine.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "rasta_safety_retransmission/sratty_sr_api_types.h"
#include "srtyp_sr_types.h"
```

Include dependency graph for srstm_sr_state_machine.h:



This graph shows which files directly or indirectly include this file:



Functions

- void `srtm_Init` (const uint32_t configured_connections)
Initialize SafRetL state machine module.
- void `srtm_ProcessConnectionStateMachine` (const uint32_t connection_id, const `srtyp_ConnectionEvents` event, const bool sequence_number_in_seq, const bool confirmed_time_stamp_in_seq)
Process RaSTA connection state machine.
- `srtyp_ConnectionStates srtm_GetConnectionState` (const uint32_t connection_id)
Return the state of a RaSTA connection state machine.

6.31.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

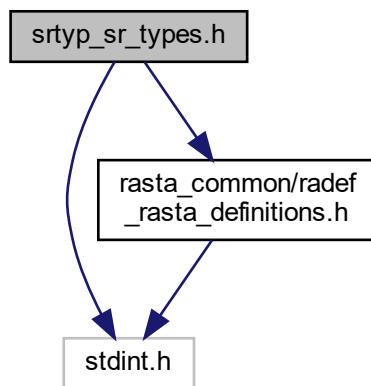
e6f89b9b2c785cdd2a74219db6d61bc80685ff37

Changelog

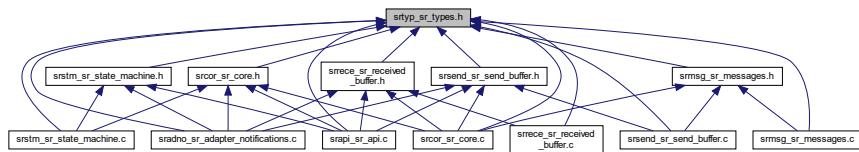
--: Initial version (-, -)

6.32 srtyp_sr_types.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
Include dependency graph for srtyp_sr_types.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct [srtyp_SrMessageHeader](#)
Typedef for a SafRetL PDU message header.
- struct [srtyp_SrMessageHeaderCreate](#)
Typedef for SafRetL PDU message header data parameters for creating a message.
- struct [srtyp_SrMessageHeaderUpdate](#)
Typedef for SafRetL PDU message header data parameters for updating a message header before sending the message.
- struct [srtyp_SrMessage](#)
Typedef for a SafRetL PDU message.
- struct [srtyp_SrMessagePayload](#)
Typedef for a SafRetL PDU message payload.
- struct [srtyp_ProtocolVersion](#)
Typedef for RaSTA protocol version array.

Macros

- #define [SRTYP_PROTOCOL_VERSION_SIZE](#) (4U)
Size of RaSTA protocol version array.

Enumerations

- enum [srtyp_ConnectionEvents](#) {

 srtyp_kConnEventMin = 0, srtyp_kConnEventNone = 0, srtyp_kConnEventOpen, srtyp_kConnEventClose

 ,

 srtyp_kConnEventSendData, srtyp_kConnEventConnReqReceived, srtyp_kConnEventConnRespReceived
 , srtyp_kConnEventRetrReqReceived ,

 srtyp_kConnEventRetrRespReceived, srtyp_kConnEventDiscReqReceived, srtyp_kConnEventHbReceived
 , srtyp_kConnEventDataReceived ,

 srtyp_kConnEventRetrDataReceived, srtyp_kConnEventSendHb, srtyp_kConnEventTimeout, srtyp_kConnEventMax
 }

Enum for the events of a RaSTA connection state machine.
- enum [srtyp_SrMessageType](#) {

 srtyp_kSrMessageMin = 6200U, srtyp_kSrMessageConnReq = 6200U, srtyp_kSrMessageConnResp = 6201U, srtyp_kSrMessageRetrReq = 6212U ,

 srtyp_kSrMessageRetrResp = 6213U, srtyp_kSrMessageDiscReq = 6216U, srtyp_kSrMessageHb = 6220U
 , srtyp_kSrMessageData = 6240U ,

 srtyp_kSrMessageRetrData = 6241U, srtyp_kSrMessageMax = 6300U }

Enum for SafRetL PDU message types.

6.32.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

e6f89b9b2c785cdd2a74219db6d61bc80685ff37

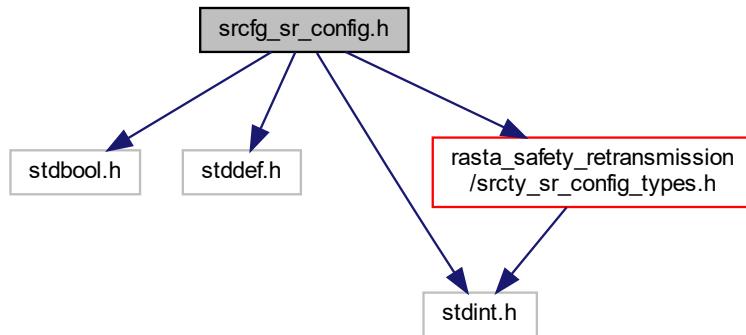
Changelog :: Initial version (-, -)

6.33 srcfg_sr_config.h File Reference

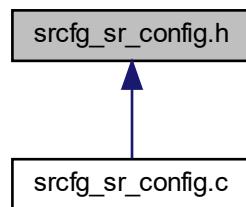
Interface of RaSTA SafRetL configuration.

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
```

Include dependency graph for srcfg_sr_config.h:



This graph shows which files directly or indirectly include this file:



Variables

- const `srcty_SafetyRetransmissionConfiguration` **safety_retransmission_configuration**
Configuration data of a safety and retransmission layer.

6.33.1 Detailed Description

Interface of RaSTA SafRetL configuration.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

ba293ba6efefd175e9347cbdc8b80d3f122e3f0d

Changelog

--: Initial version (-, -)

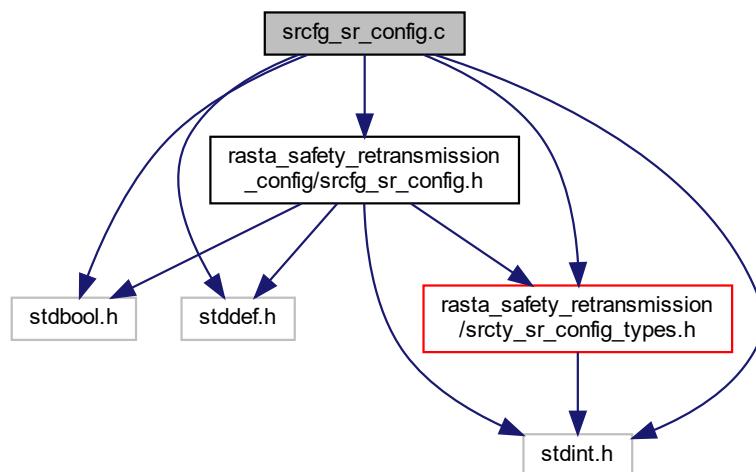
This module provides the configuration data structure for the SafRetL.

6.34 srcfg_sr_config.c File Reference

Definition of RaSTA SafRetL configuration.

```
#include "rasta_safety_retransmission_config/srcfg_sr_config.h"
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_safety_retransmission/srcty_sr_config_types.h"
```

Include dependency graph for srcfg_sr_config.c:



Variables

- const **srcty_SafetyRetransmissionConfiguration** **safety_retransmission_configuration**
Configuration data of a safety and retransmission layer.

6.34.1 Detailed Description

Definition of RaSTA SafRetL configuration.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Nicolas Andres, CSA Engineering AG, CH-4500 Solothurn

Version

ba293ba6efefd175e9347cbdc8b80d3f122e3f0d

Changelog :: Initial version (-, -)

Index

Adapter Interface component, 202
 sradin_CloseRedundancyChannel, 203
 sradin_Init, 204
 sradin_OpenRedundancyChannel, 204
 sradin_ReadMessage, 205
 sradin_SendMessage, 206
Adapter Notifications component, 207
 sradro_DiagnosticNotification, 208
 sradro_MessageReceivedNotification, 208
AddTimeToTimingDistribution
 Diagnostics component, 191
AddToBuffer
 Send Buffer component, 113
API component, 25
 srapi_CheckTimings, 26
 srapi_CloseConnection, 27
 srapi_GetConnectionState, 28
 srapi_GetInitializationState, 30
 srapi_Init, 30
 srapi_OpenConnection, 31
 srapi_ReadData, 32
 srapi_SendData, 33
Assert component, 221
 raas_AssertNotNull, 222
 raas_AssertTrue, 224
 raas_AssertU16InRange, 226
 raas_AssertU32InRange, 228
 raas_AssertU8InRange, 230

CalculateTimeliness
 Core component, 62
CheckConfirmedSequenceNumber
 Core component, 63
CheckConfirmedTimeStamp
 Core component, 64
CheckConnectionConfigurations
 Core component, 65
CheckSequenceNumber
 Core component, 66
CheckSequenceNumberRange
 Core component, 67
CheckTimeStamp
 Core component, 68
ClearMd4ContextData
 MD4 component, 177
CloseConnection
 State Machine component, 40
CloseRedundancyChannel
 State Machine component, 41
Component Specification SIL4 Package, 23
confirmed_sequence_number_tx
 srcor_RaStaConnectionData, 250
confirmed_time_stamp_rx
 srcor_RaStaConnectionData, 250
connection_configurations
 srcty_SafetyRetransmissionConfiguration, 254
connection_id
 srcty_ConnectionConfiguration, 252
CopyTempBufferIntoConnectionBuffer
 Send Buffer component, 114
CopyToContextBuffer
 MD4 component, 177
Core component, 59
 CalculateTimeliness, 62
 CheckConfirmedSequenceNumber, 63
 CheckConfirmedTimeStamp, 64
 CheckConnectionConfigurations, 65
 CheckSequenceNumber, 66
 CheckSequenceNumberRange, 67
 CheckTimeStamp, 68
 GeneralMessageCheck, 69
 GetCurrentSequenceNumberAndIncrementNumber, 71
 IsMessageTimeoutRelated, 72
 ReceivedFlowControlCheck, 73
 SendPendingMessagesWithFlowControlAllowed, 74
 SetConnectionEvent, 75
 srcor_ClearInputBufferMessagePendingFlag, 76
 srcor_CloseRedundancyChannel, 77
 srcor_GetBufferSizeAndUtilisation, 78
 srcor_GetConnectionId, 79
 srcor_GetReceivedMessagePendingFlag, 80
 srcor_HandleRetrReq, 81
 srcor_Init, 82
 srcor_InitRaStaConnData, 83
 srcor_IsConfigurationValid, 84
 srcor_IsConnRoleServer, 85
 srcor_IsHeartbeatInterval, 87
 srcor_IsMessageTimeout, 88
 srcor_IsProtocolVersionAccepted, 89
 srcor_IsReceivedMsgPendingAndBuffersNotFull, 90
 srcor_IsRetrReqSequenceNumberAvailable, 91
 srcor_kProtocolVersion, 111
 srcor_ProcessReceivedMessage, 92
 srcor_ReceiveMessage, 94
 srcor_SendConnectionStateNotification, 96
 srcor_SendConnReqMessage, 97
 srcor_SendConnRespMessage, 98
 srcor_SendDataMessage, 99
 srcor_SendDiscReqMessage, 100
 srcor_SendHbMessage, 101
 srcor_SendPendingMessages, 103
 srcor_SendRetrReqMessage, 104
 srcor_SetDiscDetailedReason, 106
 srcor_SetReceivedMessagePendingFlag, 107
 srcor_UpdateConfirmedRxSequenceNumber, 108
 srcor_UpdateConfirmedTxSequenceNumber, 109

srcor_WriteMessagePayloadToTemporaryBuffer,
 110
 CreateNewDataMsgAndAddToTempBuffer
 Send Buffer component, 115
 CreateNewHbMsgAndAddToTempBuffer
 Send Buffer component, 116
 CreateNewRetrDataMsgAndAddToTempBuffer
 Send Buffer component, 117
 CreateNewRetrReqMsgAndAddToTempBuffer
 Send Buffer component, 118

 Definitions component, 217
 raodef_kAlreadyInitialized, 221
 raodef_kDeferQueueEmpty, 221
 raodef_kInternalError, 221
 raodef_kInvalidBufferSize, 221
 raodef_kInvalidConfiguration, 221
 raodef_kInvalidMessageCrc, 221
 raodef_kInvalidMessageMd4, 221
 raodef_kInvalidMessageSize, 221
 raodef_kInvalidMessageType, 221
 raodef_kInvalidOperationInCurrentState, 221
 raodef_kInvalidParameter, 221
 raodef_kInvalidSequenceNumber, 221
 raodef_kMax, 221
 raodef_kMin, 221
 raodef_kNoError, 221
 raodef_kNoMessageReceived, 221
 raodef_kNoMessageToSend, 221
 raodef_kNotInitialized, 221
 raodef_kReceiveBufferFull, 221
 raodef_kSendBufferFull, 221
 RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE,
 219
 raodef_RaStaReturnCode, 220
 RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE
 219
 diag_timing_distr_intervals
 srcty_SafetyRetransmissionConfiguration, 254

 Diagnostics component, 190
 AddTimeToTimingDistribution, 191
 srdia_AreDiagnosticTimingIntervalsValid, 192
 srdia_IncAddressErrorCounter, 193
 srdia_IncConfirmedSequenceNumberErrorCounter,
 194
 srdia_IncSafetyCodeErrorCounter, 195
 srdia_IncSequenceNumberErrorCounter, 196
 srdia_IncTypeErrorHandlerCounter, 197
 srdia_Init, 198
 srdia_InitConnectionDiagnostics, 199
 srdia_SendDiagnosticNotification, 200
 srdia_UpdateConnectionDiagnostics, 201

 ec_address
 sraty_ConnectionDiagnosticData, 245
 ec_csn
 sraty_ConnectionDiagnosticData, 245
 ec_safety
 sraty_ConnectionDiagnosticData, 245

 ec_sn
 sraty_ConnectionDiagnosticData, 245
 ec_type
 sraty_ConnectionDiagnosticData, 245

 FunctionF
 MD4 component, 179
 FunctionG
 MD4 component, 180
 FunctionH
 MD4 component, 180

 GeneralMessageCheck
 Core component, 69
 Get
 MD4 component, 181
 GetCurrentSequenceNumberAndIncrementNumber
 Core component, 71
 GetSafetyCodeLength
 Messages component, 143
 GetUint16FromMessage
 Messages component, 144
 GetUint32FromMessage
 Messages component, 146

 Helper component, 231
 rahlp_IsU16InRange, 232
 rahlp_IsU32InRange, 233

 IncrementReceivedBufferIndexAndHandleOverflow
 Received Buffer component, 133
 IncrementSendBufferIndexAndHandleOverflow
 Send Buffer component, 119
 InitBuffer
 Send Buffer component, 120
 IsMessageTimeoutRelated
 Core component, 72
 IsMessageTypeValid
 Messages component, 147

 kMd4FunctionF
 srmd4_sr_md4.c, 301
 kMd4FunctionG
 srmd4_sr_md4.c, 301
 kMd4FunctionH
 srmd4_sr_md4.c, 301
 kMsgDetailedInfosDisconnectPosition
 srmsg_sr_messages.c, 307

 Logger component, 235
 ralog_ENABLE_LOGGER, 235

 m_w_a
 srcty_SafetyRetransmissionConfiguration, 255
 MD4 component, 175
 ClearMd4ContextData, 177
 CopyToContextBuffer, 177
 FunctionF, 179
 FunctionG, 180
 FunctionH, 180

Get, 181
Md4Body, 182
Md4Final, 183
Md4Update, 184
Set, 185
SetContextBuffer, 186
srmd4_CalculateMd4, 187
Step, 188
WriteU32toByteArray, 189
md4_initial_value
 srcty_SafetyRetransmissionConfiguration, 255
Md4Body
 MD4 component, 182
Md4Context, 240
Md4Final
 MD4 component, 183
Md4Function
 srmd4_sr_md4.c, 301
Md4Update
 MD4 component, 184
Messages component, 141
 GetSafetyCodeLength, 143
 GetUint16FromMessage, 144
 GetUint32FromMessage, 146
 IsMessageTypeValid, 147
 SetMessageHeaderInMessage, 148
 SetPayloadDataInMessage, 149
 SetProtocolVersionInMessage, 150
 SetUint16InMessage, 151
 SetUint32InMessage, 152
 SetUint64InMessage, 153
 srmsg_CheckMessage, 154
 srmsg_CreateConnReqMessage, 155
 srmsg_CreateConnRespMessage, 157
 srmsg_CreateDataMessage, 159
 srmsg_CreateDiscReqMessage, 161
 srmsg_CreateHeartbeatMessage, 162
 srmsg_CreateRetrDataMessage, 163
 srmsg_CreateRetrReqMessage, 164
 srmsg_CreateRetrRespMessage, 165
 srmsg_GetConnMessageData, 166
 srmsg_GetDataMessagePayload, 167
 srmsg_GetDiscMessageData, 169
 srmsg_GetMessageHeader, 170
 srmsg_GetMessageSequenceNumber, 171
 srmsg_GetMessageType, 172
 srmsg_Init, 173
 srmsg_UpdateMessageHeader, 174

n_diag_window
 srcty_SafetyRetransmissionConfiguration, 255
n_diagnosis
 radef_TransportChannelDiagnosticData, 242
 sraty_RedundancyChannelDiagnosticData, 247
n_max_packet
 srcty_SafetyRetransmissionConfiguration, 256
n_missed
 radef_TransportChannelDiagnosticData, 242
 sraty_RedundancyChannelDiagnosticData, 247

n_send_max
 srcty_SafetyRetransmissionConfiguration, 256
Notifications component, 35
 srnot_ConnectionStateNotification, 36
 srnot_MessageReceivedNotification, 37
 srnot_RedDiagnosticNotification, 37
 srnot_SrDiagnosticNotification, 38
number_of_connections
 srcty_SafetyRetransmissionConfiguration, 256

ProcessReceivedMessage
 State Machine component, 42
ProcessStateClosedEvents
 State Machine component, 44
ProcessStateDownEvents
 State Machine component, 45
ProcessStateRetransRequestEvents
 State Machine component, 46
ProcessStateRetransRunningEvents
 State Machine component, 47
ProcessStateStartEvents
 State Machine component, 48
ProcessStateUpEvents
 State Machine component, 49

raas_AssertNotNull
 Assert component, 222
raas_AssertTrue
 Assert component, 224
raas_AssertU16InRange
 Assert component, 226
raas_AssertU32InRange
 Assert component, 228
raas_AssertU8InRange
 Assert component, 230
raas_rasta_assert.c, 272
raas_rasta_assert.h, 265
radef_kAlreadyInitialized
 Definitions component, 221
radef_kDeferQueueEmpty
 Definitions component, 221
radef_kInternalError
 Definitions component, 221
radef_kInvalidBufferSize
 Definitions component, 221
radef_kInvalidConfiguration
 Definitions component, 221
radef_kInvalidMessageCrc
 Definitions component, 221
radef_kInvalidMessageMd4
 Definitions component, 221
radef_kInvalidMessageSize
 Definitions component, 221
radef_kInvalidMessageType
 Definitions component, 221
radef_kInvalidOperationInCurrentState
 Definitions component, 221
radef_kInvalidParameter
 Definitions component, 221

radef_kInvalidSequenceNumber
 Definitions component, 221
 radef_kMax
 Definitions component, 221
 radef_kMin
 Definitions component, 221
 radef_kNoError
 Definitions component, 221
 radef_kNoMessageReceived
 Definitions component, 221
 radef_kNoMessageToSend
 Definitions component, 221
 radef_kNotInitialized
 Definitions component, 221
 radef_kReceiveBufferFull
 Definitions component, 221
 radef_kSendBufferFull
 Definitions component, 221
 RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE
 Definitions component, 219
 radef_rasta_definitions.h, 266
 radef_RaStaReturnCode
 Definitions component, 220
 RADEF_SR_LAYER_APPLICATION_MESSAGE_LENGTH_SIZE
 Definitions component, 219
 radef_TransportChannelDiagnosticData, 241
 n_diagnosis, 242
 n_missed, 242
 t_drift, 242
 t_drift2, 242
 rahlp_IsU16InRange
 Helper component, 232
 rahlp_IsU32InRange
 Helper component, 233
 rahlp_rasta_helper.c, 273
 rahlp_rasta_helper.h, 269
 ralog_ENABLE_LOGGER
 Logger component, 235
 ralog_rasta_logger.c, 274
 ralog_rasta_logger.h, 270
 RaSTA Common Sub-Package, 217
 RaSTA Safety and Retransmission Layer Sub-Package,
 24
 rasta_network_id
 srcty_SafetyRetransmissionConfiguration, 256
 rasys_FatalError
 System Adapter component, 237
 rasys_GetRandomNumber
 System Adapter component, 238
 rasys_GetTimerGranularity
 System Adapter component, 239
 rasys_GetTimerValue
 System Adapter component, 239
 rasys_rasta_system_adapter.h, 271
 receive_buffer_free
 sraty_BufferUtilisation, 243
 receive_buffer_used
 sraty_BufferUtilisation, 243
 Received Buffer component, 132
 IncrementReceivedBufferIndexAndHandleOver-
 flow, 133
 srrece_AddToBuffer, 134
 srrece_GetFreeBufferEntries, 135
 srrece_GetPayloadSizeOfNextMessageToRead,
 136
 srrece_GetUsedBufferEntries, 137
 srrece_Init, 138
 srrece_InitBuffer, 139
 srrece_ReadFromBuffer, 140
 ReceivedFlowControlCheck
 Core component, 73
 receiver_id
 srcty_ConnectionConfiguration, 252
 Safety and Retransmission Layer API Types, 212
 sraty_ConnectionStates, 213
 sraty_DiscReason, 214
 sraty_kConnectionClosed, 213
 sraty_kConnectionDown, 213
 sraty_kConnectionMax, 213
 sraty_kConnectionMin, 213
 sraty_kConnectionNotInitialized, 213
 sraty_kConnectionRetransRequest, 213
 sraty_kConnectionRetransRunning, 213
 sraty_kConnectionStart, 213
 sraty_kConnectionUp, 213
 sraty_kDiscReasonMax, 214
 sraty_kDiscReasonMin, 214
 sraty_kDiscReasonNotInUse, 214
 sraty_kDiscReasonProtocolSequenceError, 214
 sraty_kDiscReasonProtocolVersionError, 214
 sraty_kDiscReasonRetransmissionFailed, 214
 sraty_kDiscReasonSequenceNumberError, 214
 sraty_kDiscReasonServiceNotAllowed, 214
 sraty_kDiscReasonTimeout, 214
 sraty_kDiscReasonUnexpectedMessage, 214
 sraty_kDiscReasonUserRequest, 214
 Safety and Retransmission Layer Configuration, 210
 srcty_kMinFreeEntriesReceivedBufferForReceive,
 212
 srcty_kMinFreeEntriesSendBufferForRetr, 212
 srcty_kSafetyCodeTypeFullMd4, 212
 srcty_kSafetyCodeTypeLowerMd4, 212
 srcty_kSafetyCodeTypeMax, 212
 srcty_kSafetyCodeTypeMin, 212
 srcty_kSafetyCodeTypeNone, 212
 srcty_SafetyCodeType, 211
 Safety and Retransmission Layer Types, 214
 srtyp_ConnectionEvents, 215
 srtyp_kConnEventClose, 216
 srtyp_kConnEventConnReqReceived, 216
 srtyp_kConnEventConnRespReceived, 216
 srtyp_kConnEventDataReceived, 216
 srtyp_kConnEventDiscReqReceived, 216
 srtyp_kConnEventHbReceived, 216
 srtyp_kConnEventMax, 216
 srtyp_kConnEventMin, 216

srtyp_kConnEventNone, 216
srtyp_kConnEventOpen, 216
srtyp_kConnEventRetrDataReceived, 216
srtyp_kConnEventRetrReqReceived, 216
srtyp_kConnEventRetrRespReceived, 216
srtyp_kConnEventSendData, 216
srtyp_kConnEventSendHb, 216
srtyp_kConnEventTimeout, 216
srtyp_kSrMessageConnReq, 216
srtyp_kSrMessageConnResp, 216
srtyp_kSrMessageData, 216
srtyp_kSrMessageDiscReq, 216
srtyp_kSrMessageHb, 216
srtyp_kSrMessageMax, 216
srtyp_kSrMessageMin, 216
srtyp_kSrMessageRetrData, 216
srtyp_kSrMessageRetrReq, 216
srtyp_kSrMessageRetrResp, 216
srtyp_SrMessageType, 216
safety_code_type
 srcty_SafetyRetransmissionConfiguration, 257
Send Buffer component, 111
 AddToBuffer, 113
 CopyTempBufferIntoConnectionBuffer, 114
 CreateNewDataMsgAndAddToTempBuffer, 115
 CreateNewHbMsgAndAddToTempBuffer, 116
 CreateNewRetrDataMsgAndAddToTempBuffer,
 117
 CreateNewRetrReqMsgAndAddToTempBuffer, 118
 IncrementSendBufferIndexAndHandleOverflow,
 119
 InitBuffer, 120
 srsend_AddToBuffer, 121
 srsend_GetFreeBufferEntries, 122
 srsend_GetNumberOfMessagesToSend, 123
 srsend_GetUsedBufferEntries, 124
 srsend_Init, 125
 srsend_InitBuffer, 126
 srsend_IsSequenceNumberInBuffer, 127
 srsend_PreparesBufferForRetr, 128
 srsend_ReadMessageToSend, 130
 srsend_RemoveFromBuffer, 131
send_buffer_free
 sraty_BufferUtilisation, 244
send_buffer_used
 sraty_BufferUtilisation, 244
sender_id
 srcty_ConnectionConfiguration, 252
SendPendingMessagesWithFlowControlAllowed
 Core component, 74
Set
 MD4 component, 185
SetConnectionEvent
 Core component, 75
SetContextBuffer
 MD4 component, 186
SetMessageHeaderInMessage
 Messages component, 148
SetPayloadDataInMessage
 Messages component, 149
SetProtocolVersionInMessage
 Messages component, 150
SetUint16InMessage
 Messages component, 151
SetUint32InMessage
 Messages component, 152
SetUint64InMessage
 Messages component, 153
sradin_CloseRedundancyChannel
 Adapter Interface component, 203
sradin_Init
 Adapter Interface component, 204
sradin_OpenRedundancyChannel
 Adapter Interface component, 204
sradin_ReadMessage
 Adapter Interface component, 205
sradin_SendMessage
 Adapter Interface component, 206
sradin_sr_adapter_interface.h, 275
sradno_DiagnosticNotification
 Adapter Notifications component, 208
sradno_MessageReceivedNotification
 Adapter Notifications component, 208
sradno_sr_adapter_notifications.c, 285
sradno_sr_adapter_notifications.h, 277
srapi_CheckTimings
 API component, 26
srapi_CloseConnection
 API component, 27
srapi_GetConnectionState
 API component, 28
srapi_GetInitializationState
 API component, 30
srapi_Init
 API component, 30
srapi_OpenConnection
 API component, 31
srapi_ReadData
 API component, 32
srapi_SendData
 API component, 33
srapi_sr_api.c, 286
srapi_sr_api.h, 278
sraty_BufferUtilisation, 243
 receive_buffer_free, 243
 receive_buffer_used, 243
 send_buffer_free, 244
 send_buffer_used, 244
sraty_ConnectionDiagnosticData, 244
 ec_address, 245
 ec_csn, 245
 ec_safety, 245
 ec_sn, 245
 ec_type, 245
 t_alive_distribution, 246
 t_rtd_distribution, 246

sraty_ConnectionStates
 Safety and Retransmission Layer API Types, 213
 sraty_DiscReason
 Safety and Retransmission Layer API Types, 214
 sraty_kConnectionClosed
 Safety and Retransmission Layer API Types, 213
 sraty_kConnectionDown
 Safety and Retransmission Layer API Types, 213
 sraty_kConnectionMax
 Safety and Retransmission Layer API Types, 213
 sraty_kConnectionMin
 Safety and Retransmission Layer API Types, 213
 sraty_kConnectionNotInitialized
 Safety and Retransmission Layer API Types, 213
 sraty_kConnectionRetransRequest
 Safety and Retransmission Layer API Types, 213
 sraty_kConnectionRetransRunning
 Safety and Retransmission Layer API Types, 213
 sraty_kConnectionStart
 Safety and Retransmission Layer API Types, 213
 sraty_kConnectionUp
 Safety and Retransmission Layer API Types, 213
 sraty_kDiscReasonMax
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonMin
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonNotInUse
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonProtocolSequenceError
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonProtocolVersionError
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonRetransmissionFailed
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonSequenceNumberError
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonServiceNotAllowed
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonTimeout
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonUnexpectedMessage
 Safety and Retransmission Layer API Types, 214
 sraty_kDiscReasonUserRequest
 Safety and Retransmission Layer API Types, 214
 sraty_RedundancyChannelDiagnosticData, 246
 n_diagnosis, 247
 n_missed, 247
 t_drift, 247
 t_drift2, 247
 transport_channel_id, 247
 sraty_sr_api_types.h, 280
 srcfg_sr_config.c, 322
 srcfg_sr_config.h, 321
 srcor_ClearInputBufferMessagePendingFlag
 Core component, 76
 srcor_CloseRedundancyChannel
 Core component, 77
 srcor_GetBufferSizeAndUtilisation
 Core component, 78
 srcor_GetConnectionId
 Core component, 79
 srcor_GetReceivedMessagePendingFlag
 Core component, 80
 srcor_HandleRetrReq
 Core component, 81
 srcor_Init
 Core component, 82
 srcor_InitRaStaConnData
 Core component, 83
 srcor_InputBuffer, 248
 srcor_IsConfigurationValid
 Core component, 84
 srcor_IsConnRoleServer
 Core component, 85
 srcor_IsHeartbeatInterval
 Core component, 87
 srcor_IsMessageTimeout
 Core component, 88
 srcor_IsProtocolVersionAccepted
 Core component, 89
 srcor_IsReceivedMsgPendingAndBuffersNotFull
 Core component, 90
 srcor_IsRetrReqSequenceNumberAvailable
 Core component, 91
 srcor_kProtocolVersion
 Core component, 111
 srcor_ProcessReceivedMessage
 Core component, 92
 srcor_RaStaConnectionData, 249
 confirmed_sequence_number_tx, 250
 confirmed_time_stamp_rx, 250
 t_alive, 250
 t_rtd, 250
 srcor_ReceiveMessage
 Core component, 94
 srcor_SendConnectionStateNotification
 Core component, 96
 srcor_SendConnReqMessage
 Core component, 97
 srcor_SendConnRespMessage
 Core component, 98
 srcor_SendDataMessage
 Core component, 99
 srcor_SendDiscReqMessage
 Core component, 100
 srcor_SendHbMessage
 Core component, 101
 srcor_SendPendingMessages
 Core component, 103
 srcor_SendRetrReqMessage
 Core component, 104
 srcor_SetDiscDetailedReason
 Core component, 106
 srcor_SetReceivedMessagePendingFlag
 Core component, 107
 srcor_sr_core.c, 288

srcor_sr_core.h, 291
srcor_TemporaryBuffer, 251
srcor_UpdateConfirmedRxSequenceNumber
 Core component, 108
srcor_UpdateConfirmedTxSequenceNumber
 Core component, 109
srcor_WriteMessagePayloadToTemporaryBuffer
 Core component, 110
srcty_ConnectionConfiguration, 251
 connection_id, 252
 receiver_id, 252
 sender_id, 252
srcty_kMinFreeEntriesReceivedBufferForReceive
 Safety and Retransmission Layer Configuration,
 212
srcty_kMinFreeEntriesSendBufferForRetr
 Safety and Retransmission Layer Configuration,
 212
srcty_kSafetyCodeTypeFullMd4
 Safety and Retransmission Layer Configuration,
 212
srcty_kSafetyCodeTypeLowerMd4
 Safety and Retransmission Layer Configuration,
 212
srcty_kSafetyCodeTypeMax
 Safety and Retransmission Layer Configuration,
 212
srcty_kSafetyCodeTypeMin
 Safety and Retransmission Layer Configuration,
 212
srcty_kSafetyCodeTypeNone
 Safety and Retransmission Layer Configuration,
 212
srcty_Md4InitValue, 253
srcty_SafetyCodeType
 Safety and Retransmission Layer Configuration,
 211
srcty_SafetyRetransmissionConfiguration, 253
 connection_configurations, 254
 diag_timing_distr_intervals, 254
 m_w_a, 255
 md4_initial_value, 255
 n_diag_window, 255
 n_max_packet, 256
 n_send_max, 256
 number_of_connections, 256
 rasta_network_id, 256
 safety_code_type, 257
 t_h, 257
 t_max, 257
srcty_sr_config_types.c, 294
srcty_sr_config_types.h, 281
srdia_AreDiagnosticTimingIntervalsValid
 Diagnostics component, 192
srdia_IncAddressErrorCounter
 Diagnostics component, 193
srdia_IncConfirmedSequenceNumberErrorCounter
 Diagnostics component, 194
srdia_IncSafetyCodeErrorCounter
 Diagnostics component, 195
srdia_IncSequenceNumberErrorCounter
 Diagnostics component, 196
srdia_IncTypeErrorHandler
 Diagnostics component, 197
srdia_Init
 Diagnostics component, 198
srdia_InitConnectionDiagnostics
 Diagnostics component, 199
srdia_SendDiagnosticNotification
 Diagnostics component, 200
srdia_sr_diagnostics.c, 295
srdia_sr_diagnostics.h, 297
srdia_SrConnectionDiagnostics, 258
srdia_UpdateConnectionDiagnostics
 Diagnostics component, 201
srmd4_CalculateMd4
 MD4 component, 187
srmd4_Md4, 258
srmd4_sr_md4.c, 299
 kMd4FunctionF, 301
 kMd4FunctionG, 301
 kMd4FunctionH, 301
 Md4Function, 301
srmd4_sr_md4.h, 302
srmsg_CheckMessage
 Messages component, 154
srmsg_CreateConnReqMessage
 Messages component, 155
srmsg_CreateConnRespMessage
 Messages component, 157
srmsg_CreateDataMessage
 Messages component, 159
srmsg_CreateDiscReqMessage
 Messages component, 161
srmsg_CreateHeartbeatMessage
 Messages component, 162
srmsg_CreateRetrDataMessage
 Messages component, 163
srmsg_CreateRetrReqMessage
 Messages component, 164
srmsg_CreateRetrRespMessage
 Messages component, 165
srmsg_GetConnMessageData
 Messages component, 166
srmsg_GetDataMessagePayload
 Messages component, 167
srmsg_GetDiscMessageData
 Messages component, 169
srmsg_GetMessageHeader
 Messages component, 170
srmsg_GetMessageSequenceNumber
 Messages component, 171
srmsg_GetMessageType
 Messages component, 172
srmsg_Init
 Messages component, 173

srmsg_sr_messages.c, 303
 kMsgDetailedInfosDisconnectPosition, 307
srmsg_sr_messages.h, 307
srmsg_UpdateMessageHeader
 Messages component, 174
srnot_ConnectionStateNotification
 Notifications component, 36
srnot_MessageReceivedNotification
 Notifications component, 37
srnot_RedDiagnosticNotification
 Notifications component, 37
srnot_sr_notifications.h, 283
srnot_SrDiagnosticNotification
 Notifications component, 38
srrce_AddToBuffer
 Received Buffer component, 134
srrce_GetFreeBufferEntries
 Received Buffer component, 135
srrce_GetPayloadSizeOfNextMessageToRead
 Received Buffer component, 136
srrce_GetUsedBufferEntries
 Received Buffer component, 137
srrce_Init
 Received Buffer component, 138
srrce_InitBuffer
 Received Buffer component, 139
srrce_ReadFromBuffer
 Received Buffer component, 140
srrce_sr_received_buffer.c, 309
srrce_sr_received_buffer.h, 311
SrReceivedBuffer, 259
srsend_AddToBuffer
 Send Buffer component, 121
srsend_GetFreeBufferEntries
 Send Buffer component, 122
srsend_GetNumberOfMessagesToSend
 Send Buffer component, 123
srsend_GetUsedBufferEntries
 Send Buffer component, 124
srsend_Init
 Send Buffer component, 125
srsend_InitBuffer
 Send Buffer component, 126
srsend_IsSequenceNumberInBuffer
 Send Buffer component, 127
srsend_PreparesBufferForRetr
 Send Buffer component, 128
srsend_ReadMessageToSend
 Send Buffer component, 130
srsend_RemoveFromBuffer
 Send Buffer component, 131
srsend_send_buffers
srsend_sr_send_buffer.c, 314
srsend_sr_send_buffer.h, 312
 srsend_send_buffers, 314
SrSendBuffer, 260
SrSendMessage, 261
srstm_GetConnectionState
 State Machine component, 51
srstm_Init
 State Machine component, 52
srstm_ProcessConnectionStateMachine
 State Machine component, 53
srstm_sr_state_machine.c, 316
srstm_sr_state_machine.h, 318
srtyp_ConnectionEvents
 Safety and Retransmission Layer Types, 215
srtyp_kConnEventClose
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventConnReqReceived
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventConnRespReceived
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventDataReceived
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventDiscReqReceived
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventHbReceived
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventMax
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventMin
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventNone
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventOpen
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventRetrDataReceived
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventRetrReqReceived
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventRetrRespReceived
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventSendData
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventSendHb
 Safety and Retransmission Layer Types, 216
srtyp_kConnEventTimeout
 Safety and Retransmission Layer Types, 216
srtyp_kSrMessageConnReq
 Safety and Retransmission Layer Types, 216
srtyp_kSrMessageConnResp
 Safety and Retransmission Layer Types, 216
srtyp_kSrMessageData
 Safety and Retransmission Layer Types, 216
srtyp_kSrMessageDiscReq
 Safety and Retransmission Layer Types, 216
srtyp_kSrMessageHb
 Safety and Retransmission Layer Types, 216
srtyp_kSrMessageMax
 Safety and Retransmission Layer Types, 216
srtyp_kSrMessageMin
 Safety and Retransmission Layer Types, 216
srtyp_kSrMessageRetrData
 Safety and Retransmission Layer Types, 216

srtyp_kSrMessageRetrReq
 Safety and Retransmission Layer Types, 216

srtyp_kSrMessageRetrResp
 Safety and Retransmission Layer Types, 216

srtyp_ProtocolVersion, 261
 version, 262

srtyp_sr_types.h, 319

srtyp_SrMessage, 262

srtyp_SrMessageHeader, 262

srtyp_SrMessageHeaderCreate, 263

srtyp_SrMessageHeaderUpdate, 264

srtyp_SrMessagePayload, 264

srtyp_SrMessageType
 Safety and Retransmission Layer Types, 216

StartRetransmission
 State Machine component, 55

State Machine component, 39
 CloseConnection, 40
 CloseRedundancyChannel, 41
 ProcessReceivedMessage, 42
 ProcessStateClosedEvents, 44
 ProcessStateDownEvents, 45
 ProcessStateRetransRequestEvents, 46
 ProcessStateRetransRunningEvents, 47
 ProcessStateStartEvents, 48
 ProcessStateUpEvents, 49
 srstm_GetConnectionState, 51
 srstm_Init, 52
 srstm_ProcessConnectionStateMachine, 53
 StartRetransmission, 55
 UpdateConnectionState, 57
 UpdateConnectionStateWithDiscReason, 58

Step
 MD4 component, 188

System Adapter component, 236
 rasys_FatalError, 237
 rasys_GetRandomNumber, 238
 rasys_GetTimerGranularity, 239
 rasys_GetTimerValue, 239

t_alive
 srcor_RaStaConnectionData, 250

t_alive_distribution
 sraty_ConnectionDiagnosticData, 246

t_drift
 radef_TransportChannelDiagnosticData, 242
 sraty_RedundancyChannelDiagnosticData, 247

t_drift2
 radef_TransportChannelDiagnosticData, 242
 sraty_RedundancyChannelDiagnosticData, 247

t_h
 srcty_SafetyRetransmissionConfiguration, 257

t_max
 srcty_SafetyRetransmissionConfiguration, 257

t_rtd
 srcor_RaStaConnectionData, 250

t_rtd_distribution
 sraty_ConnectionDiagnosticData, 246

transport_channel_id

sraty_RedundancyChannelDiagnosticData, 247

UpdateConnectionState
 State Machine component, 57

UpdateConnectionStateWithDiscReason
 State Machine component, 58

version
 srtyp_ProtocolVersion, 262

WriteU32toByteArray
 MD4 component, 189