RaSTA Protocol Reference Stack

Attachment of Software Component Design Specification
Basic Integrity

Project:	RaSTA Protocol Reference Stack	**Customer:**	SBB
Author:	R. Schenk	**Version:**	V1.1.0
Document-Id:	SBB-RaSTA-066	**Status:**	Release
Date:	20/12/2022	**Filename:**	SBB-RaSTA-066-AttachmentOfSoftwareComponentDesignSpecification-BasicIntegrity.pdf

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* [9]
- Module: Synonym for component
- Member: Function or enumeration which is part of a component
- Class: Describes a structure (struct)

This terms are defined by doxygen and cannot be adjusted.

1.2 Introduction

This document aims to specify the implementation of the individual components of the RaSTA protocol stack. The Component Specification Basic Integrity Package contains all component specifications of the following layers with basic integrity:

- RaSTA Redundancy Layer

The RaSTA Common Sub-Package (including Definitions component, Assert component, Helper component, Logger component and System Adapter component) is included because it is used by the RaSTA Redundancy Layer Sub-Package components. This package is reviewed and verified in the *SBB-RaSTA-071 - Attachment of Software Component Design Specification SIL4*, including the here found documentation and traceability about it.

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.↩ 0.0 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 (In this case its only the requirements from the RaSTA Common Sub-Package package). 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 Basic Integrity 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 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 (-, -)

Module red_types

-: Initial version (-, -)

File redcfg_red_config.c

-: Initial version (-, -)

File redcfg_red_config.h

-: Initial version (-, -)

File redcor_red_core.c

-: Initial version (-, -)

File redcor_red_core.h

-: Initial version (-, -)

File redcrc_red_crc.c

-: Initial version (-, -)

File redcrc_red_crc.h

-: Initial version (-, -)

File redcty_red_config_types.c

-: Initial version (-, -)

File redcty_red_config_types.h

-: Initial version (-, -)

File reddfq_red_defer_queue.c

-: Initial version (-, -)

File reddfq_red_defer_queue.h

-: Initial version (-, -)

File reddia_red_diagnostics.c

-: Initial version (-, -)

File reddia_red_diagnostics.h

-: Initial version (-, -)

File redint_red_interface.c

-: Initial version (-, -)

File redint_red_interface.h

-: Initial version (-, -)

File redmsg_red_messages.c

-: Initial version (-, -)

File redmsg_red_messages.h

-: Initial version (-, -)

File rednot_red_notifications.h

-: Initial version (-, -)

File redrbf_red_received_buffer.c

-: Initial version (-, -)

File redrbf_red_received_buffer.h

-: Initial version (-, -)

File redstm_red_state_machine.c

-: Initial version (-, -)

File redstm_red_state_machine.h

-: Initial version (-, -)

File redtri_transport_interface.h

-: Initial version (-, -)

File redtrn_transport_notifications.c

-: Initial version (-, -)

File redtrn_transport_notifications.h

-: Initial version (-, -)

3 Requirement Traceability Summary

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 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 drift2

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

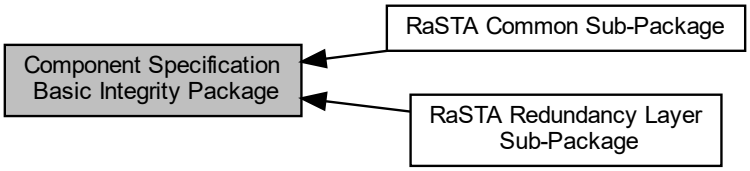
RASW-410 Get Timer Value Function Structure

RASW-422 Timer Value

4 Module Documentation

4.1 Component Specification Basic Integrity Package

Collaboration diagram for Component Specification Basic Integrity Package:



Modules

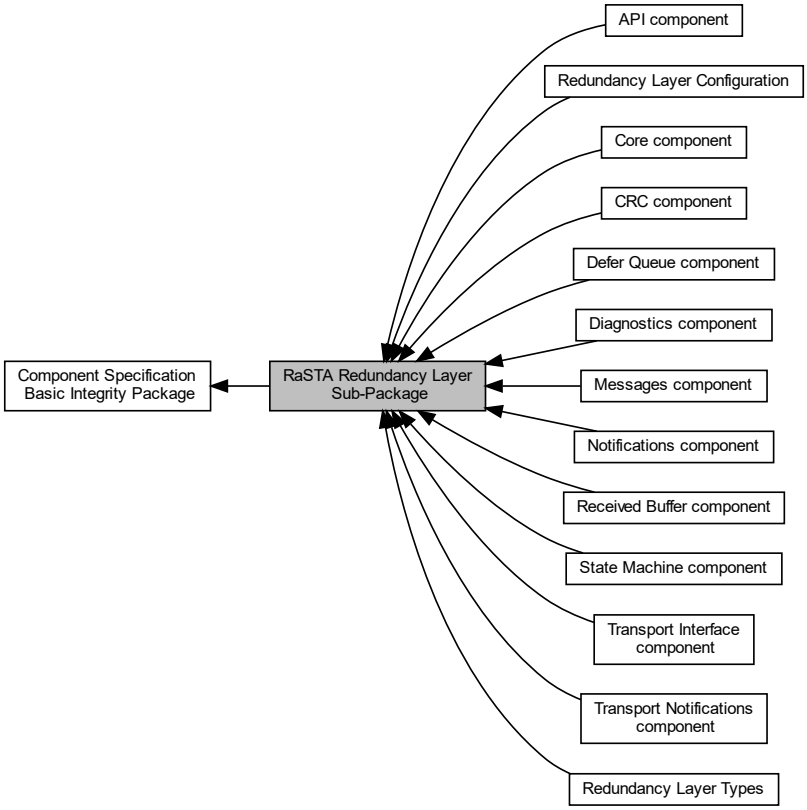
- RaSTA Redundancy Layer Sub-Package
- RaSTA Common Sub-Package

4.1.1 Detailed Description

Specification of all components of the RaSTA Stack with basic integrity.

4.2 RaSTA Redundancy Layer Sub-Package

Collaboration diagram for RaSTA Redundancy Layer Sub-Package:



Modules

- API component

  Interface of RaSTA redundancy layer.
- Notifications component

  Interface of RaSTA RedL notifications to the SafRetL adapter.
- State Machine component

  Interface of RaSTA redundancy layer state machine module.
- Core component

  Interface of RaSTA redundancy layer core module.
- Defer Queue component

  Interface of RaSTA redundancy layer defer queue module.
- Received Buffer component

  Interface of RaSTA redundancy layer received messages buffer module.
- Messages component

  Interface of RaSTA redundancy layer messages module.
- CRC component

  Interface of RaSTA redundancy layer CRC module.
- Diagnostics component

Interface of RaSTA redundancy layer diagnostics.

- Transport Interface component

  Interface of RaSTA transport layer.
- Transport Notifications component

  Interface of RaSTA transport layer notifications.
- Redundancy Layer Configuration

  Type definitions of RaSTA redundancy layer configuration.
- Redundancy Layer Types

  Internal type definitions of RaSTA redundancy layer.

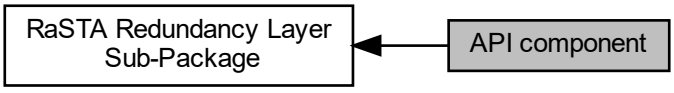
4.2.1 Detailed Description

Specification of all components within the RaSTA Redundancy Layer.

4.3 API component

Interface of RaSTA redundancy layer.

Collaboration diagram for API component:



Functions

- radef_RaStaReturnCode redint_Init (const redcty_RedundancyLayerConfiguration ∗const redundancy_layer_configuration)

  Initialize the RedL.
- radef_RaStaReturnCode redint_GetInitializationState (void)

  Get the initialization state of the RedL.
- radef_RaStaReturnCode redint_OpenRedundancyChannel (const uint32_t redundancy_channel_id)

  Open a given redundancy channel.
- radef_RaStaReturnCode redint_CloseRedundancyChannel (const uint32_t redundancy_channel_id)

  Close a given redundancy channel.
- radef_RaStaReturnCode redint_SendMessage (const uint32_t redundancy_channel_id, const uint16_t message_size, const uint8_t ∗const message_data)

  Send a message over a given redundancy channel.
- radef_RaStaReturnCode redint_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 given redundancy channel.
- radef_RaStaReturnCode redint_CheckTimings (void)

  Check redundancy layer timings and read pending messages form the transport channels.

4.3.1 Detailed Description

Interface of RaSTA redundancy layer.

Specification of the API component of the RaSTA Redundancy Layer.

This module defines and implements the RedL interface functions:

- Initialize the RedL
- Get initialization state of the RedL
- Open redundancy channel
- Close redundancy channel
- Send data
- Read data
- Check timings

4.3.2 Function Documentation

4.3.2.1 redint_CheckTimings() `radef_RaStaReturnCode redint_CheckTimings ( void )`

Check redundancy layer timings and read pending messages form the transport channels.

This function calls the internal function ReceivedMessagesPolling() and checks the defer queue timeout by calling reddfq_IsTimeout() and redstm_ProcessChannelStateMachine() with the redstm_kRedundancyChannelEventDeferTimeout event, for all configured redundancy channels which are in the redstm_kRedundancyChannelStateUp state.

Precondition

The RedL interface module must be initialized, otherwise a radef_kNotInitialized error is returned.

Remarks

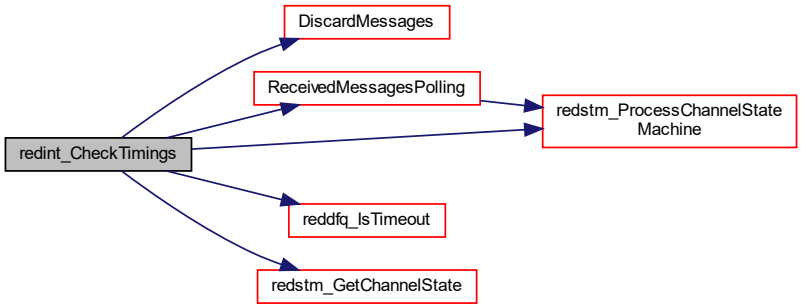
This function must be called periodically, in a appropriate interval related to the configured timings.

Returns

radef_kNoError -> successful operation

radef_kNotInitialized -> module not initialized

Here is the call graph for this function:



4.3.2.2 redint_CloseRedundancyChannel() `radef_RaStaReturnCode redint_CloseRedundancyChannel ( const uint32_t redundancy_channel_id )`

Close a given redundancy channel.

This function closes a given redundancy channel by calling redstm_ProcessChannelStateMachine() with the redstm_kRedundancyChannelEventClose event.

Precondition

The RedL interface module must be initialized, otherwise a radef_kNotInitialized error is returned.

Parameters

in	*redundancy_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.

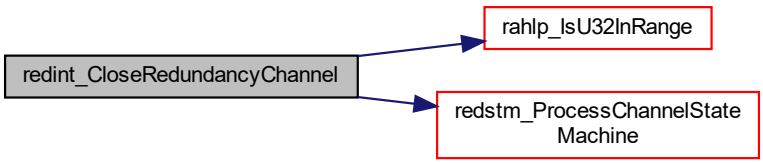
Returns

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 redint_GetInitializationState() `radef_RaStaReturnCode redint_GetInitializationState ( void )`

Get the initialization state of the RedL.

This function returns the initialization state of the RedL.

Returns

radef_kNoError -> successful initialized

radef_kNotInitialized -> not initialized

4.3.2.4 redint_Init() `radef_RaStaReturnCode redint_Init ( const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration )`

Initialize the RedL.

This function is used to initialize the red_interface module. It checks the passed redcty_RedundancyLayerConfiguration configuration and if it is valid, it initializes the internal data, the state machine module, the defer queue module, the received buffer module and the core module.

Precondition

The RedL interface module must not be initialized, otherwise a radef_kAlreadyInitialized error is returned.

Parameters

in	*redundancy_layer_configuration*	Pointer to RedL configuration. More details about valid configuration can be found directly in redcty_RedundancyLayerConfiguration. If the pointer is NULL, a radef_kInvalidParameter error is returned.

Returns

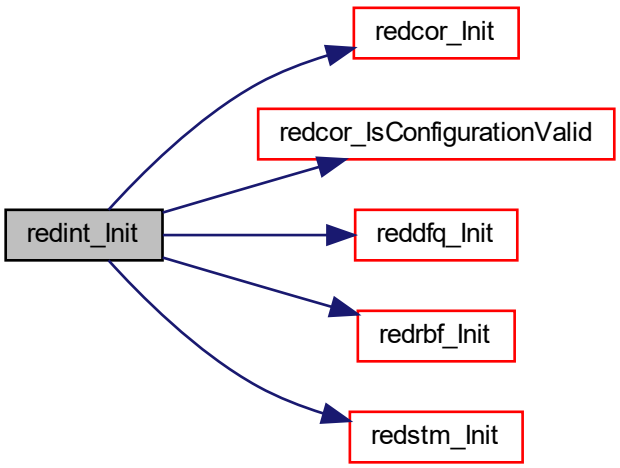
radef_kNoError -> successful initialized

radef_kAlreadyInitialized -> module already initialized

radef_kInvalidConfiguration -> invalid configuration data

radef_kInvalidParameter -> invalid parameter

Here is the call graph for this function:



4.3.2.5 redint_OpenRedundancyChannel() `radef_RaStaReturnCode redint_OpenRedundancyChannel ( const uint32_t redundancy_channel_id )`

Open a given redundancy channel.

This function initializes the internal data of a given redundancy channel and opens a redundancy channel by calling redstm_ProcessChannelStateMachine() with the redstm_kRedundancyChannelEventOpen event.

Precondition

The RedL interface module must be initialized, otherwise a radef_kNotInitialized error is returned.

Parameters

in	*redundancy_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.

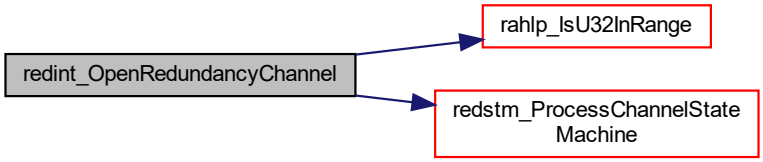
Returns

radef_kNoError -> successful operation

radef_kNotInitialized -> module not initialized

radef_kInvalidParameter -> invalid parameter

Here is the call graph for this function:



4.3.2.6 redint_ReadMessage()

```
radef_RaStaReturnCode redint_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 given redundancy channel.

This function reads a received message from the received buffer of a given redundancy channel. If no message is in the received buffer, a radef_kNoMessageReceived error is returned.

Precondition

The RedL interface module must be initialized, otherwise a radef_kNotInitialized error is returned.

Parameters

in	*redundancy_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.
in	*buffer_size*	Size of the buffer [bytes] available in the upper layer. Valid range: RADEF_SR_LAYER_MESSAGE_HEADER_SIZE <= value <= RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE. If the size of the message to read is greater then the buffer size a radef_kInvalidBufferSize error is returned.
out	*message_size*	Pointer to the size of the received message data [bytes]. If the pointer is NULL, a radef_kInvalidParameter error is returned.
out	*message_buffer*	Pointer to a buffer for saving the received message. If the pointer is NULL, a radef_kInvalidParameter error is returned.

Returns

radef_kNoError -> successful operation

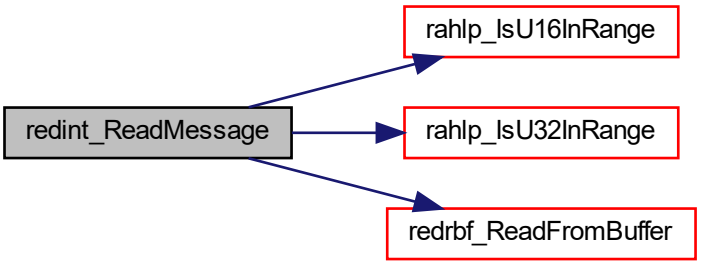
radef_kNoMessageReceived -> no message received (used for polling)

radef_kNotInitialized -> module not initialized

radef_kInvalidParameter -> invalid parameter

radef_kInvalidBufferSize -> invalid buffer size

Here is the call graph for this function:



4.3.2.7 redint_SendMessage()

```
radef_RaStaReturnCode redint_SendMessage (
        const uint32_t redundancy_channel_id,
        const uint16_t message_size,
        const uint8_t *const message_data )
```

Send a message over a given redundancy channel.

This function copies the given message to the send buffer of the given redundancy channel and triggers the sending of this message by calling redstm_ProcessChannelStateMachine() with the redstm_kRedundancyChannelEventSendData event.

Precondition

The RedL interface module must be initialized, otherwise a radef_kNotInitialized error is returned.

Parameters

in	*redundancy_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.
in	*message_size*	The size of the message data [bytes] must be in the range from RADEF_SR_LAYER_MESSAGE_HEADER_SIZE to RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidMessageSize error is returned.
in	*message_data*	Pointer to the message data byte array which must be send. If the pointer is NULL, a radef_kInvalidParameter error is returned. For the message data the full value range is valid and usable.

Returns

radef_kNoError -> successful operation

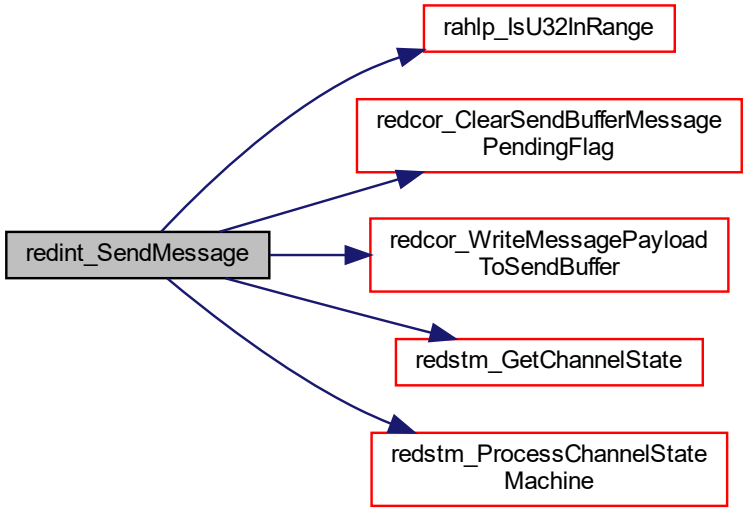
radef_kNotInitialized -> module not initialized

radef_kInvalidParameter -> invalid parameter

radef_kInvalidMessageSize -> invalid message size

radef_kInvalidOperationInCurrentState -> sending messages is not allowed in the current redundancy channel state machine state

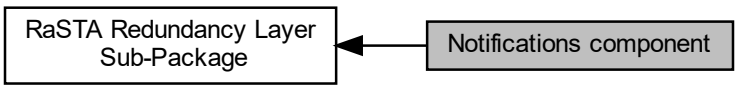
Here is the call graph for this function:



4.4 Notifications component

Interface of RaSTA RedL notifications to the SafRetL adapter.

Collaboration diagram for Notifications component:



Functions

- void rednot_MessageReceivedNotification (const uint32_t red_channel_id)

  Redundancy layer message received notification function to SafRetL adapter.
- void rednot_DiagnosticNotification (const uint32_t red_channel_id, const uint32_t tr_channel_id, const radef_TransportChannelDiagnosticData TransportChannelDiagnosticData)

  Redundancy layer diagnostic notification function to SafRetL adapter.

4.4.1 Detailed Description

Interface of RaSTA RedL notifications to the SafRetL adapter.

Specification of the Notifications component of the RaSTA Redundancy Layer.

This module defines the message received notification and diagnostic notification function interfaces for the RedL. The RedL only defines the interface, the implementation of this notification functions must be done in the SafRetL adapter.

4.4.2 Function Documentation

4.4.2.1 rednot_DiagnosticNotification()

```
void rednot_DiagnosticNotification (
        const uint32_t red_channel_id,
        const uint32_t tr_channel_id,
        const radef_TransportChannelDiagnosticData TransportChannelDiagnosticData )
```

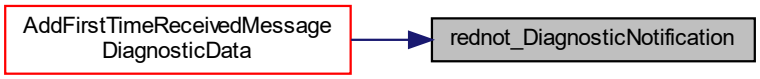
Redundancy layer diagnostic notification function to SafRetL adapter.

This function is called by the redundancy layer to notify the upper layer about new diagnostic data from a specific redundancy channel.

Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.
in	*tr_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS.
in	*TransportChannelDiagnosticData*	Transport channel diagnostic data structure. Structure and valid ranges can be found in radef_TransportChannelDiagnosticData.

Here is the caller graph for this function:



4.4.2.2 rednot_MessageReceivedNotification() `void rednot_MessageReceivedNotification ( const uint32_t red_channel_id )`

Redundancy layer message received notification function to SafRetL adapter.

This function is called by the redundancy layer to notify the upper layer that a received message is ready to be read on a specific redundancy channel.

Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.

Here is the caller graph for this function:

4.5 State Machine component

Interface of RaSTA redundancy layer state machine module.

Collaboration diagram for State Machine component:

Enumerations

- enum redstm_RedundancyChannelStates {
  redstm_kRedundancyChannelStateMin = 0U , redstm_kRedundancyChannelStateNotInitialized = 0U ,
  redstm_kRedundancyChannelStateClosed , redstm_kRedundancyChannelStateUp ,
  redstm_kRedundancyChannelStateMax }

  Enum for the states of a redundancy channel state machine.
- enum redstm_RedundancyChannelEvents {
  redstm_kRedundancyChannelEventMin = 0U , redstm_kRedundancyChannelEventOpen = 0U , redstm_kRedundancyChannelE
  , redstm_kRedundancyChannelEventReceiveData ,
  redstm_kRedundancyChannelEventSendData , redstm_kRedundancyChannelEventDeferTimeout ,
  redstm_kRedundancyChannelEventMax }

  Enum for the events of a redundancy channel state machine.

Functions

- static void ProcessStateClosedEvents (const uint32_t red_channel_id, const redstm_RedundancyChannelEvents event)

  Process events in the closed state.
- static void ProcessStateUpEvents (const uint32_t red_channel_id, const redstm_RedundancyChannelEvents event)

  Process events in the up state.
- void redstm_Init (const uint32_t configured_red_channels)

  Initialize the RedL state machine module.
- void redstm_ProcessChannelStateMachine (const uint32_t red_channel_id, const redstm_RedundancyChannelEvents event)

  Process redundancy channel state machine.
- redstm_RedundancyChannelStates redstm_GetChannelState (const uint32_t red_channel_id)

  Return the state of a redundancy channel state machine.

4.5.1 Detailed Description

Interface of RaSTA redundancy layer state machine module.

Specification of the State Machine component of the RaSTA Redundancy Layer.

This module implements the redundancy layer state machine which handles the closed and up states of the redundancy channels.

4.5.2 Enumeration Type Documentation

4.5.2.1 redstm_RedundancyChannelEvents `enum redstm_RedundancyChannelEvents`

Enum for the events of a redundancy channel state machine.

Enumerator

redstm_kRedundancyChannelEventMin	Min value for redundancy channel events enum.
redstm_kRedundancyChannelEventOpen	open redundancy channel event
redstm_kRedundancyChannelEventClose	close redundancy channel event
redstm_kRedundancyChannelEventReceiveData	receive data event
redstm_kRedundancyChannelEventSendData	send data event
redstm_kRedundancyChannelEventDeferTimeout	defer queue timeout event

4.5.2.2 redstm_RedundancyChannelStates `enum redstm_RedundancyChannelStates`

Enum for the states of a redundancy channel state machine.

Enumerator

redstm_kRedundancyChannelStateMin	Min value for redundancy channel state enum.
redstm_kRedundancyChannelStateNotInitialized	not initialized
redstm_kRedundancyChannelStateClosed	redundancy channel closed
redstm_kRedundancyChannelStateUp	redundancy channel up
redstm_kRedundancyChannelStateMax	Max value for redundancy channel state enum.

4.5.3 Function Documentation

4.5.3.1 ProcessStateClosedEvents() `static void ProcessStateClosedEvents ( const uint32_t red_channel_id, const redstm_RedundancyChannelEvents event ) [static]`

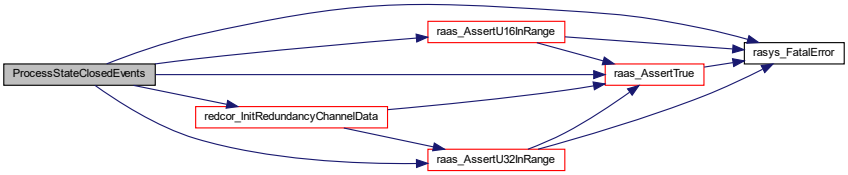
Process events in the closed state.

This function processes the events in the closed state of the redundancy channel state machine, calls the appropriate functions and does the appropriate state transitions. This function checks at the beginning if the closed state is active, else a radef_kInternalError fatal error message is thrown.

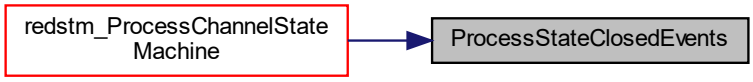
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*event*	Event to process. All enum entries of redstm_RedundancyChannelEvents are valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.2 ProcessStateUpEvents()

```
static void ProcessStateUpEvents (
        const uint32_t red_channel_id,
        const redstm_RedundancyChannelEvents event )  [static]
```

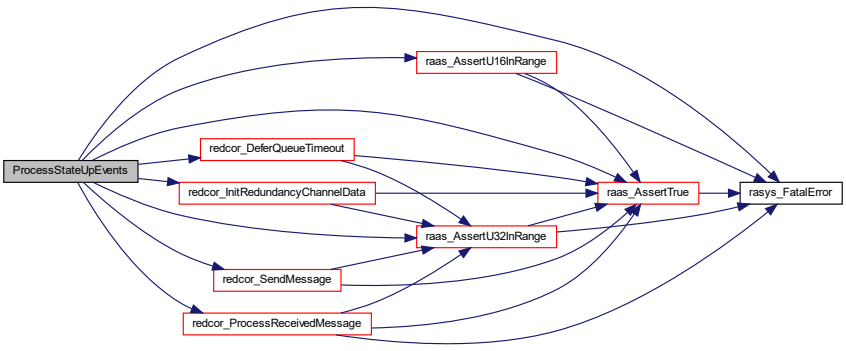
Process events in the up state.

This function processes the events in the up state of the redundancy channel state machine, calls the appropriate functions and does the appropriate state transitions. This function checks at the beginning if the up state is active, else a radef_kInternalError fatal error message is thrown.

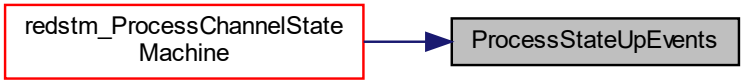
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*event*	Event to process. All enum entries of redstm_RedundancyChannelEvents are valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.3 redstm_GetChannelState() `redstm_RedundancyChannelStates redstm_GetChannelState ( const uint32_t red_channel_id )`

Return the state of a redundancy channel state machine.

Precondition

The state machine module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

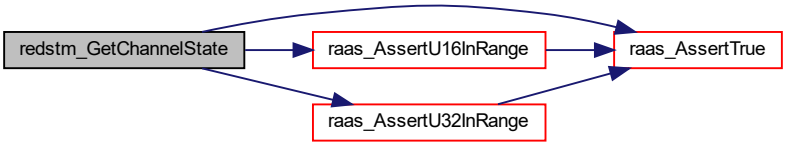
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

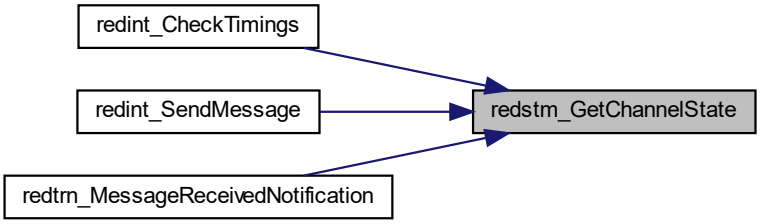
Returns

State of the redundancy channel state machine. All enum entries of redstm_RedundancyChannelStates are valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.4 redstm_Init() `void redstm_Init (`

`const uint32_t` *`configured_red_channels`* `)`

Initialize the RedL state machine module.

This function is used to initialize the state machine module. It saves the passed number of redundancy channels. For all configured channels the state machine is properly initialized. A fatal error is raised, if this function is called multiple times.

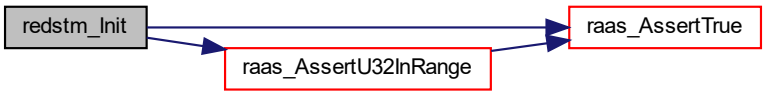
Precondition

The state machine module must not be initialized, otherwise a radef_kAlreadyInitialized fatal error is thrown.

Parameters

in	*configured_red_channels*	Number of configured redundancy channels. Valid range: 1 <= value <= RADEF_MAX_NUMBER_OF_RED_CHANNELS.

Here is the call graph for this function:



Here is the caller graph for this function:

4.5.3.5 redstm_ProcessChannelStateMachine()

```
void redstm_ProcessChannelStateMachine (
        const uint32_t red_channel_id,
        const redstm_RedundancyChannelEvents event )
```

Process redundancy channel state machine.

This function processes the events of the redundancy channel state machine, calls the appropriate functions and does the appropriate state transitions. All details can be found in the image below or in figure 14 & 15 of the standard "Elektrische Bahn-Signalanlagen – Teil 200: Sicheres Übertragungsprotokoll RaSTA nach DIN EN 50159 (VDE0831-159); Juni 2015":

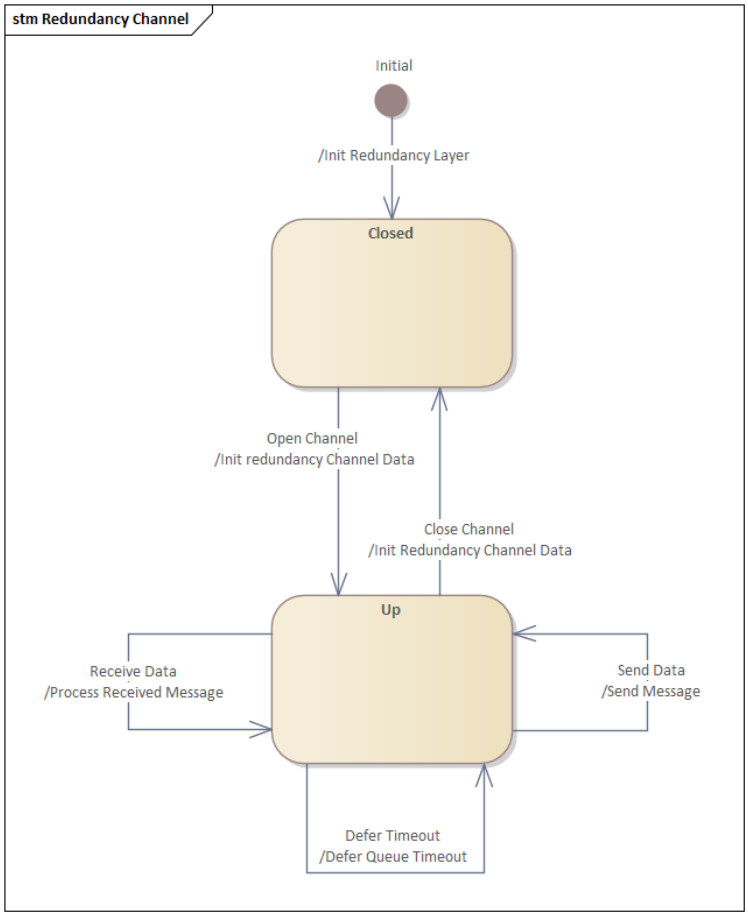


Figure 1 RaSTA Redundancy Channel State Machine

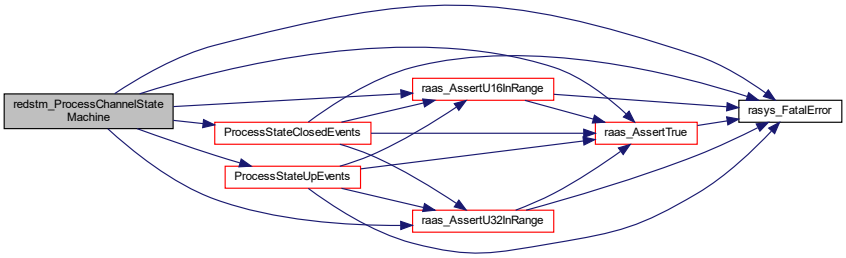
Precondition

The state machine module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

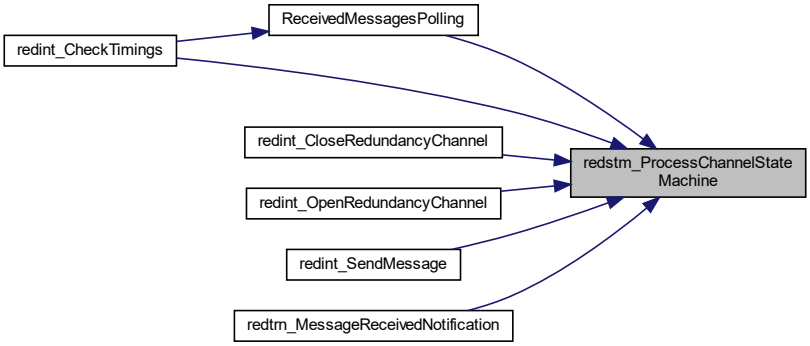
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*event*	Event to process. All enum entries of redstm_RedundancyChannelEvents are valid and usable.

Here is the call graph for this function:



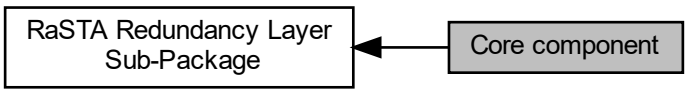
Here is the caller graph for this function:



4.6 Core component

Interface of RaSTA redundancy layer core module.

Collaboration diagram for Core component:



Classes

- struct redcor_InputBuffer

  Struct for the newly received message input buffer.
- struct redcor_SendBuffer

  Struct for the message payload send buffer.
- struct redcor_RedundancyChannelData

  Struct for the process data of a redundancy channel.

Functions

- static void DeliverDeferQueue (const uint32_t red_channel_id)

  This function delivers the messages from the defer queue to the received buffer.
- static void AddMessageToReceivedBufferAndDeliverDeferQueue (const uint32_t red_channel_id)

  Add a received message to the received buffer and call the DeliverDeferQueue() function.
- bool redcor_IsConfigurationValid (const redcty_RedundancyLayerConfiguration ∗const redundancy_layer_configuration)

  Checks if the redundancy layer configuration is valid.
- void redcor_Init (const redcty_RedundancyLayerConfiguration ∗const redundancy_layer_configuration)

  Initialize all data of the redundancy layer core.
- void redcor_InitRedundancyChannelData (const uint32_t red_channel_id)

  Initialize the data of a dedicated redundancy channel.
- void redcor_DeferQueueTimeout (const uint32_t red_channel_id)

  Handle the defer queue timeout and deliver the defer queue messages to the received buffer.
- void redcor_WriteReceivedMessageToInputBuffer (const uint32_t red_channel_id, const uint32_t transport↩ _channel_id, const redtyp_RedundancyMessage ∗const received_message)

  Write a received message to the input buffer.
- void redcor_ClearInputBufferMessagePendingFlag (const uint32_t red_channel_id)

  Clear input buffer message pending flag.
- void redcor_ProcessReceivedMessage (const uint32_t red_channel_id)

  Process a received message from the input buffer.
- void redcor_SetMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Set a flag, which indicates that a received message is pending to read from the transport layer.
- bool redcor_GetMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Get the received message pending flag for a dedicated transport channel.
- void redcor_ClearMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Clear the received message pending flag.
- void redcor_WriteMessagePayloadToSendBuffer (const uint32_t red_channel_id, const uint16_t payload_↩ size, const uint8_t ∗const payload_data)

Write message payload to send buffer.

- void redcor_ClearSendBufferMessagePendingFlag (const uint32_t red_channel_id)

  Clear send buffer message pending flag.

- void redcor_SendMessage (const uint32_t red_channel_id)

  Send a redundancy layer message from the send buffer to the transport channels.

- void redcor_GetAssociatedRedundancyChannel (const uint32_t transport_channel_id, uint32_t *const red_channel_id)

  Get the associated redundancy channel from a given transport channel.

4.6.1 Detailed Description

Interface of RaSTA redundancy layer core module.

Specification of the Core component of the RaSTA Redundancy Layer.

This module provides the core functionality of the RaSTA redundancy layer. This includes:

- Check of redundancy layer configuration data
- Defer queue timeout handling and defer queue delivery to the received buffer
- Processing of received redundancy layer messages and forwarding to the received buffer
- Send redundancy layer messages to the transport channels

4.6.2 Function Documentation

4.6.2.1 AddMessageToReceivedBufferAndDeliverDeferQueue() `static void AddMessageToReceivedBufferAndDeliverDeferQueue ( const uint32_t red_channel_id ) [static]`

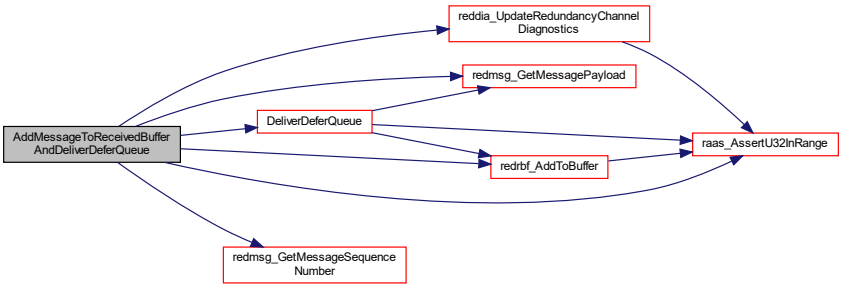
Add a received message to the received buffer and call the DeliverDeferQueue() function.

This function adds a in sequence received message from the input buffer to the received buffer, updates the diagnostics, increases SeqRx and calls the function DeliverDeferQueue().

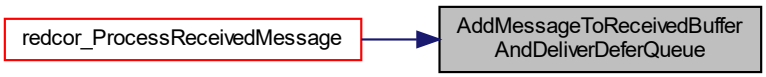
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.2 DeliverDeferQueue() `static void DeliverDeferQueue ( const uint32_t red_channel_id ) [static]`

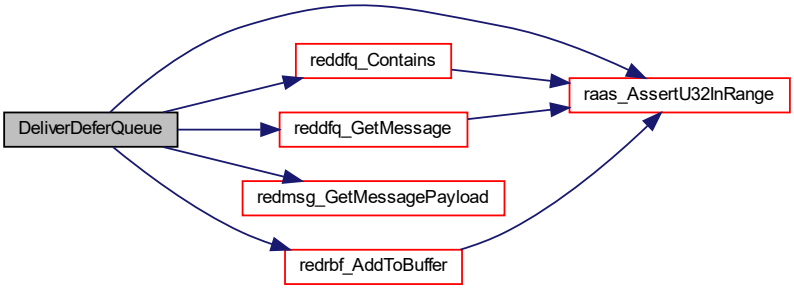
This function delivers the messages from the defer queue to the received buffer.

This function delivers the messages stored in the defer queue to the received buffer, as long as there is no sequence number missing. The sequence number receive is updated for each delivered message.

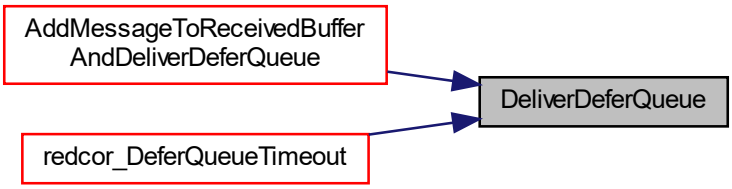
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.3 redcor_ClearInputBufferMessagePendingFlag()

```
void redcor_ClearInputBufferMessagePending↩
Flag (
            const uint32_t red_channel_id )
```

Clear input buffer message pending flag.

This function clears the input buffer message pending flag in a given redundancy channel.

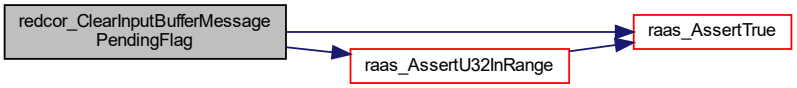
Precondition

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

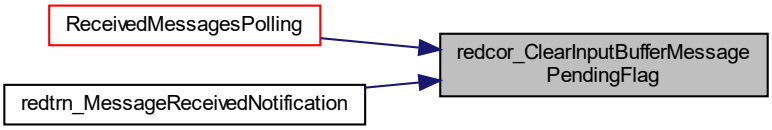
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.4 redcor_ClearMessagePendingFlag()

```
void redcor_ClearMessagePendingFlag (
        const uint32_t red_channel_id,
        const uint32_t transport_channel_id )
```

Clear the received message pending flag.

This function clears the received message pending flag in a given redundancy channel for a given transport channel.

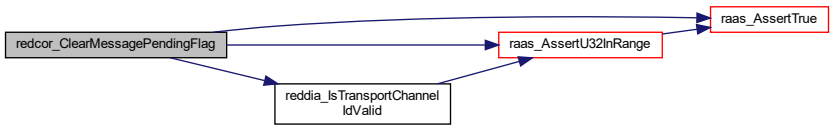
Precondition

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

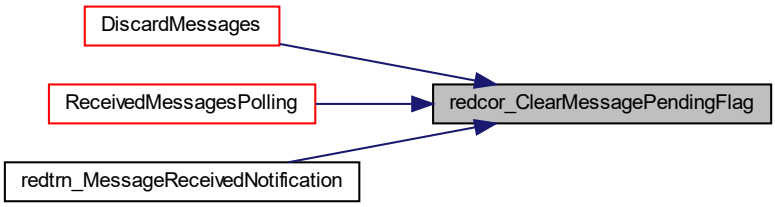
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the specified redundancy channel, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.5 redcor_ClearSendBufferMessagePendingFlag() `void redcor_ClearSendBufferMessagePending↩ Flag (`

```
            const uint32_t red_channel_id )
```

Clear send buffer message pending flag.

This function clears the send buffer message pending flag in a given redundancy channel.

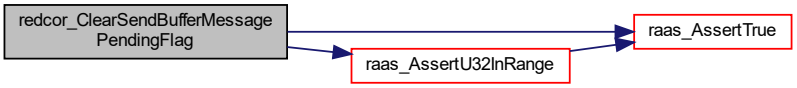
Precondition

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

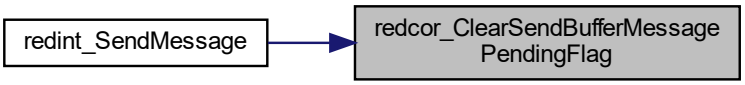
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.6 redcor_DeferQueueTimeout() `void redcor_DeferQueueTimeout ( const uint32_t red_channel_id )`

Handle the defer queue timeout and deliver the defer queue messages to the received buffer.

This function handles the defer queue timeout and delivers the defer queue messages to the received buffer. First, the sequence number receive is set to the oldest sequence number found in the defer queue. The messages stored in the defer queue are delivered to the received buffer, as long as there is no sequence number missing. The sequence number receive is updated for each delivered message.

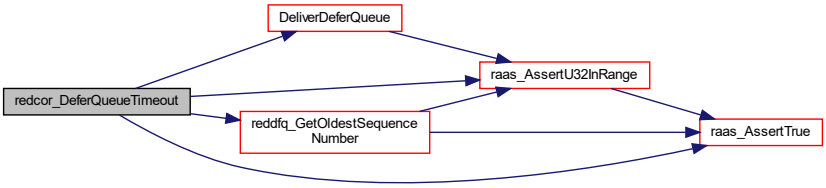
Precondition

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

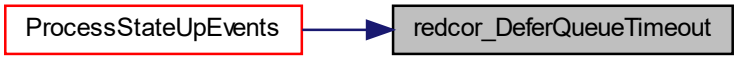
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.7 redcor_GetAssociatedRedundancyChannel()

```
void redcor_GetAssociatedRedundancyChannel (
        const uint32_t transport_channel_id,
        uint32_t *const red_channel_id )
```

Get the associated redundancy channel from a given transport channel.

This function gets the redundancy channel, which is associated to the given transport channel.

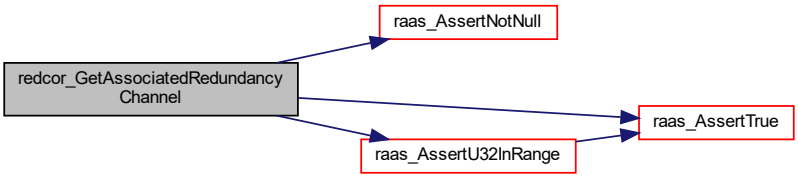
Precondition

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

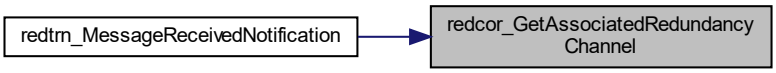
Parameters

in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the a redundancy channel, a radef_kInvalidParameter fatal error is thrown.
out	*red_channel_id*	Pointer to associated redundancy channel identification. 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:



4.6.2.8 redcor_GetMessagePendingFlag()

```
bool redcor_GetMessagePendingFlag (
        const uint32_t red_channel_id,
        const uint32_t transport_channel_id )
```

Get the received message pending flag for a dedicated transport channel.

This function gets the received message pending flag from a given redundancy channel for a given transport channel.

Precondition

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

Parameters

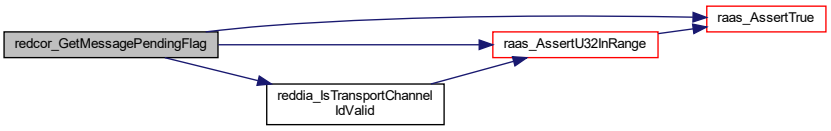
in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the specified redundancy channel, a radef_kInvalidParameter fatal error is thrown.

Returns

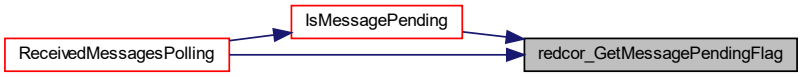
true, if a received message is pending

false, if no received message is pending

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.9 redcor_Init() `void redcor_Init (`

`const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration )`

Initialize all data of the redundancy layer core.

This function is used to initialize the core module. The validity of the configuration is checked by calling the redcor_IsConfigurationValid() function. If the configuration is not valid a radef_kInvalidConfiguration fatal error is thrown. It saves the passed redundancy layer configuration. The initialization of the redundancy messages module and the redundancy diagnostics module is called. Finally for all configured channels the redcor_InitRedundancyChannelData function is called to properly initialize the data for all configured channels.

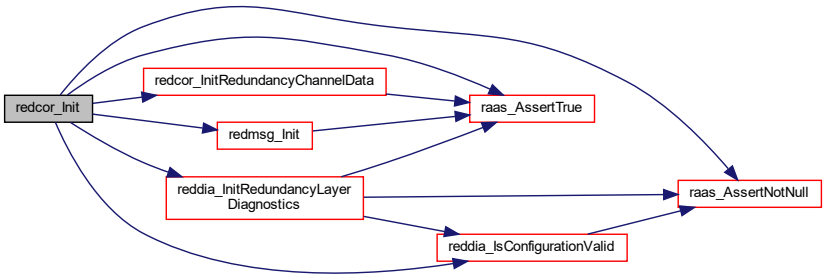
Precondition

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

Parameters

in	*redundancy_layer_configuration*	Pointer to redundancy layer configuration data structure. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown. If the configuration is not valid a::radef_kInvalidConfiguration fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:

4.6.2.10 redcor_InitRedundancyChannelData() `void redcor_InitRedundancyChannelData ( const uint32_t red_channel_id )`

Initialize the data of a dedicated redundancy channel.

This function initializes the data of a given redundancy channel. It resets the following properties:

- Sequence number receive
- Sequence number transmit
- Received data pending flags
- Input buffer
- Send buffer

The following initialization functions are also called here for the given redundancy channel:

- reddfq_InitDeferQueue()
- redrbf_InitBuffer()
- reddia_InitRedundancyChannelDiagnostics()

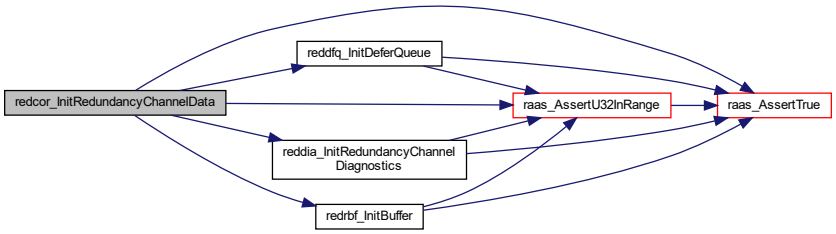
Precondition

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

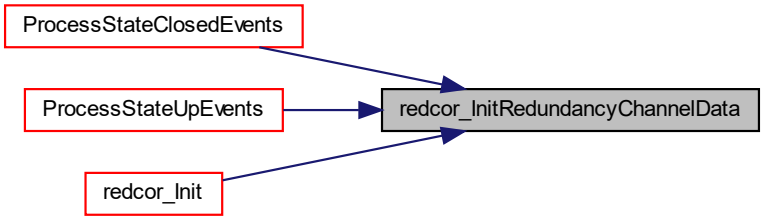
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.11 redcor_IsConfigurationValid() `bool redcor_IsConfigurationValid ( const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration )`

Checks if the redundancy layer configuration is valid.

This function checks if all elements of the redundancy layer configuration are in their valid ranges and if all configuration elements are consistent to the others. This function is only a wrapper function, which calls reddia_IsConfigurationValid(). For a more detailed description, see the description of reddia_IsConfigurationValid().

Parameters

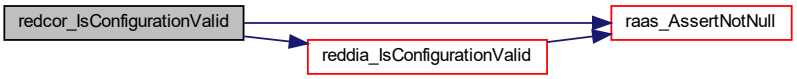
in	*redundancy_layer_configuration*	Pointer to redundancy layer configuration data structure. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

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:

4.6.2.12 redcor_ProcessReceivedMessage() `void redcor_ProcessReceivedMessage ( const uint32_t red_channel_id )`

Process a received message from the input buffer.

This function processes a received message from the input buffer. The following tasks are done:

- Check if a received message is in the input buffer, else a radef_kNoMessageReceived fatal error is thrown.
- Check if the message CRC is correct, else the message is ignored
- Check that after the initialization only one message with sequence number 0 is processed, else ignore the message
- Ignore messages with Seq_pdu $<$ Seq_rx, but update the redundancy channel diagnostics
- For messages with Seq_pdu = Seq_rx, add message to received buffer, increase Seq_rx, update the redundancy channel diagnostics
- For messages with Seq_rx $<$ Seq_pdu $<=$ (Seq_rx + N_defer_queue_size $*$ 10), add message to the defer queue, if not already in the queue, and update the redundancy channel diagnostics
- For messages with Seq_pdu $>$ (Seq_rx + N_defer_queue_size $*$ 10), ignore the message without update of the redundancy channel diagnostics
- Clear the input buffer message pending flag

All detailed flow chart can be found in figure 18 of the standard "Elektrische Bahn-Signalanlagen – Teil 200: Sicheres Übertragungsprotokoll RaSTA nach DIN EN 50159 (VDE0831-159); Juni 2015".

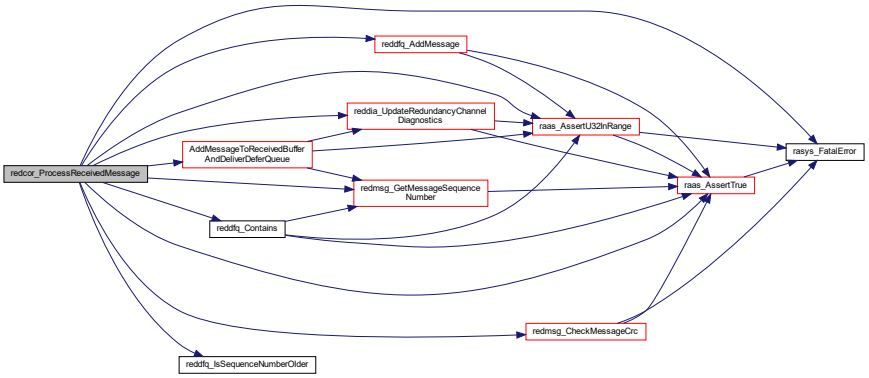
Precondition

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

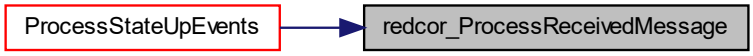
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.13 redcor_SendMessage() `void redcor_SendMessage ( const uint32_t red_channel_id )`

Send a redundancy layer message from the send buffer to the transport channels.

This function sends a redundancy layer message from the send buffer of a given redundancy channel all to the transport channels associated with this redundancy channel. The sequence number transmit is increased. The message in buffer flag is cleared after sending the messages.

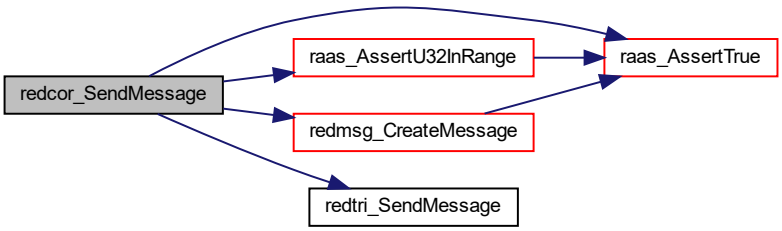
Precondition

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

Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:

4.6.2.14 redcor_SetMessagePendingFlag()

```
void redcor_SetMessagePendingFlag (
        const uint32_t red_channel_id,
        const uint32_t transport_channel_id )
```

Set a flag, which indicates that a received message is pending to read from the transport layer.

This function sets the received message pending flag in a given redundancy channel for a given transport channel.

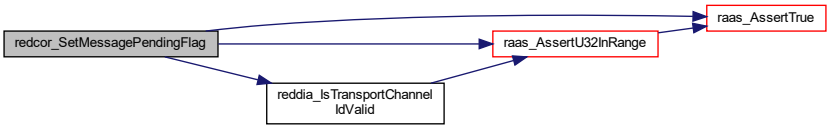
Precondition

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

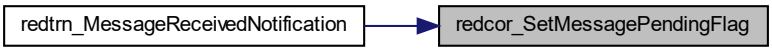
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the specified redundancy channel, a radef_kInvalidParameter fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.15 redcor_WriteMessagePayloadToSendBuffer() `void redcor_WriteMessagePayloadToSendBuffer (`

```
            const uint32_t red_channel_id,
            const uint16_t payload_size,
            const uint8_t *const payload_data )
```

Write message payload to send buffer.

This function copies a message to send and its message size to the send buffer of a given redundancy channel. The message_in_buffer flag is set.

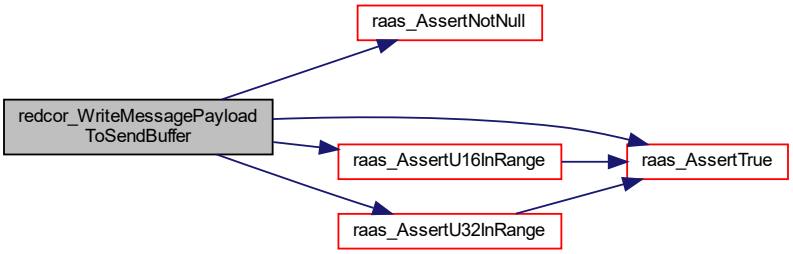
Precondition

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

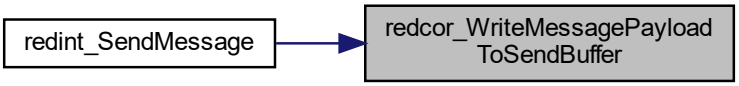
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*payload_size*	The payload_size must be in the range from RADEF_SR_LAYER_MESSAGE_HEADER_SIZE to RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidParameter fatal error message is thrown.
in	*payload_data*	Pointer to message payload data byte array which must be written to the buffer. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown. For the message payload data the full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.2.16 redcor_WriteReceivedMessageToInputBuffer()

```
void redcor_WriteReceivedMessageToInputBuffer (
            const uint32_t red_channel_id,
            const uint32_t transport_channel_id,
            const redtyp_RedundancyMessage *const received_message )
```

Write a received message to the input buffer.

This function copies a received message and its message size and transport channel identification to the input buffer. The message_in_buffer flag is set. The transport_channel_id is saved to be passed to the redundancy channel diagnostics.

Precondition

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

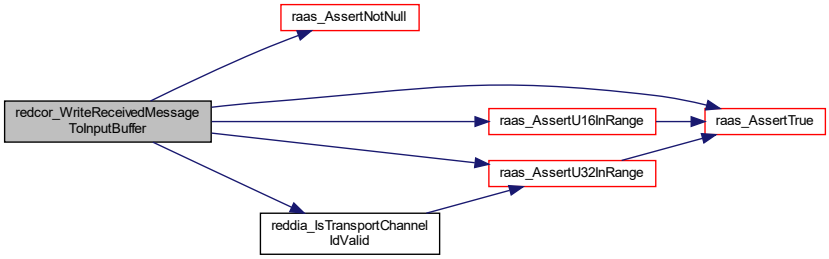
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the specified redundancy channel, a radef_kInvalidParameter fatal error is thrown.

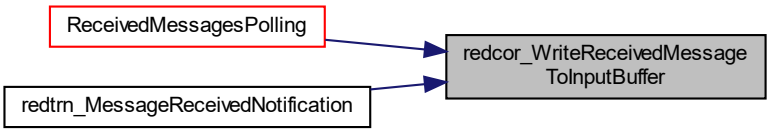
Parameters

in	*received_message*	Pointer to struct containing the message. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown. The received_message->message_size must be in the range from RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE to RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidParameter fatal error message is thrown.

Here is the call graph for this function:



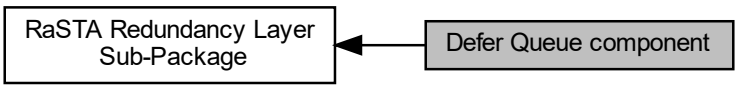
Here is the caller graph for this function:



4.7 Defer Queue component

Interface of RaSTA redundancy layer defer queue module.

Collaboration diagram for Defer Queue component:



Functions

- void reddfq_Init (const uint32_t configured_red_channels, const uint32_t configured_defer_queue_size, const uint32_t configured_t_seq)

  Initialization of the data of the RedL defer queue module.
- void reddfq_InitDeferQueue (const uint32_t red_channel_id)

  Initialization of the defer queue of a dedicated redundancy channel.
- void reddfq_AddMessage (const uint32_t red_channel_id, const redtyp_RedundancyMessage *const redundancy_message)

  Add a redundancy layer message to the defer queue. If the queue is full, the message will be ignored.
- void reddfq_GetMessage (const uint32_t red_channel_id, const uint32_t sequence_number, redtyp_RedundancyMessage *const redundancy_message)

  Get and remove a redundancy layer message from the defer queue.
- bool reddfq_IsTimeout (const uint32_t red_channel_id)

  Check defer queue timeout on a dedicated redundancy channel.
- bool reddfq_Contains (const uint32_t red_channel_id, const uint32_t sequence_number)

  Checks if a message with a defined sequence number is in the defer queue.
- uint32_t reddfq_GetOldestSequenceNumber (const uint32_t red_channel_id)

  Returns the oldest sequence number found in the defer queue.
- uint32_t reddfq_GetUsedEntries (const uint32_t red_channel_id)

  Get the number of used defer queue entries.
- bool reddfq_IsSequenceNumberOlder (const uint32_t sequence_number_to_compare, const uint32_t sequence_number_reference)

  Returns true, if the sequence_number_to_compare is older than sequence_number_reference.

4.7.1 Detailed Description

Interface of RaSTA redundancy layer defer queue module.

Specification of the Defer Queue component of the RaSTA Redundancy Layer.

This module provides the RaSTA RedL defer queue functionality. It is used to store out of sequence received messages in the defer queue. If the message with the missing sequence number is received or the Tseq timeout is reached, the core module delivers the messages form the defer queue to the received buffer. This module provides the following functionality:

- Initialize defer queue of a redundancy channel
- Add a message to a defer queue
- Get a message from a defer queue
- Check defer queue timeout
- Check if the defer queue contains a message with a defined sequence number
- Get the minimum sequence number in the defer queue
- Get the number of used defer queue entries

4.7.2 Function Documentation

4.7.2.1 reddfq_AddMessage()

```
void reddfq_AddMessage (
        const uint32_t red_channel_id,
        const redtyp_RedundancyMessage *const redundancy_message )
```

Add a redundancy layer message to the defer queue. If the queue is full, the message will be ignored.

When there is free space in the defer queue, a RedL message with a current time stamp is added to the buffer. If the defer queue is full, the message is ignored. After adding the message to the defer queue, the number of used entries is updated.

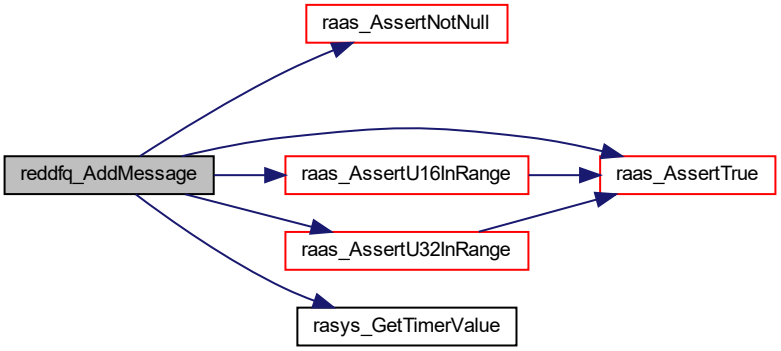
Precondition

The defer queue module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*redundancy_message*	Pointer to the message struct to add to the defer queue. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown. The redundancy_message->message_size must be in the range from RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE to RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidParameter fatal error message is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:

4.7.2.2 reddfq_Contains() `bool reddfq_Contains (`

```
const uint32_t red_channel_id,
const uint32_t sequence_number )
```

Checks if a message with a defined sequence number is in the defer queue.

This function checks if a message with a defined sequence number is in the defer queue and returns true if the sequence number was found.

Precondition

The defer queue module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

Parameters

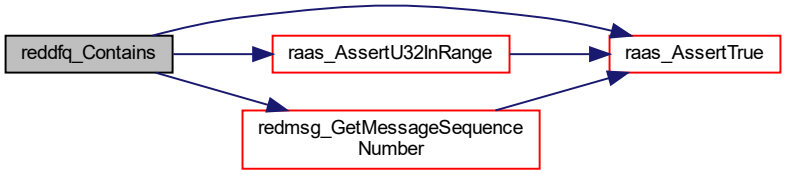
in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*sequence_number*	Sequence number of the message to search. The full value range is valid and usable.

Returns

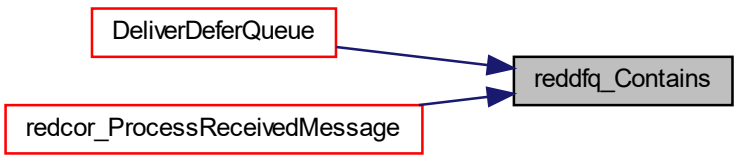
true, if a message with the sequence number is found.

false, if no message with the sequence number is found.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.3 reddfq_GetMessage()

```
void reddfq_GetMessage (
        const uint32_t red_channel_id,
        const uint32_t sequence_number,
        redtyp_RedundancyMessage *const redundancy_message )
```

Get and remove a redundancy layer message from the defer queue.

When there is messages in the defer queue with the requested sequence number, it is read from the defer queue, saved into the passed structure and the number of used entries is updated and the message size in the defer queue is set to 0 to indicate a free entry. The message size read from the defer queue must be in the range from RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE to RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE, else a radef_kInternalError fatal error message is thrown. If the message with the requested sequence number is not found in the defer queue, a radef_kInvalidSequenceNumber fatal error is thrown.

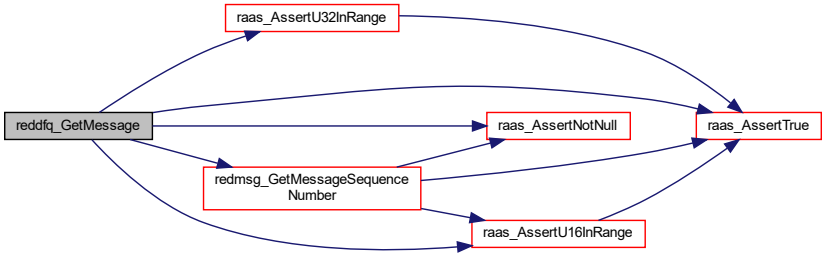
Precondition

The defer queue module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*sequence_number*	Sequence number of the message to read and remove from the queue. The full value range is valid and usable.
in	*redundancy_message*	Pointer to RedL message structure, where the message must be saved. 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:

4.7.2.4 reddfq_GetOldestSequenceNumber() `uint32_t reddfq_GetOldestSequenceNumber ( const uint32_t red_channel_id )`

Returns the oldest sequence number found in the defer queue.

This function searches the oldest sequence number of all messages in the defer queue and returns it. This function takes respect to uint32_t wrap around for up counted sequence numbers. A sequence number is detected to be older, if the difference of two sequence numbers - 1 is < (UINT32_MAX / 2).

Precondition

The defer queue module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

The defer queue module must not be empty, otherwise a radef_kDeferQueueEmpty fatal error is thrown.

The number of used defer queue entries must be in the range form 1 to the configured defer queue size, otherwise a radef_kInternalError fatal error is thrown.

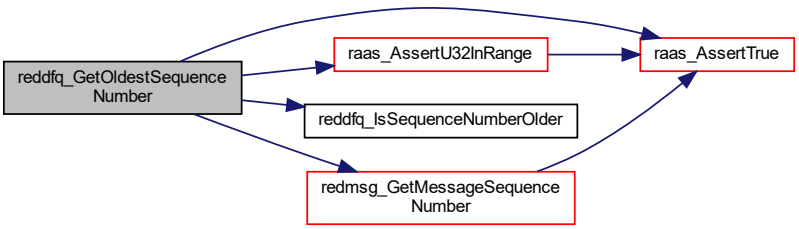
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

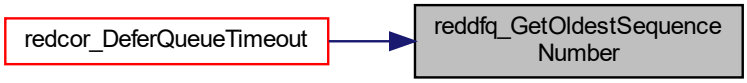
Returns

Oldest sequence number found in the defer queue.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.5 reddfq_GetUsedEntries() `uint32_t reddfq_GetUsedEntries ( const uint32_t red_channel_id )`

Get the number of used defer queue entries.

This function returns the number of used defer queue entries.

Precondition

The defer queue module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

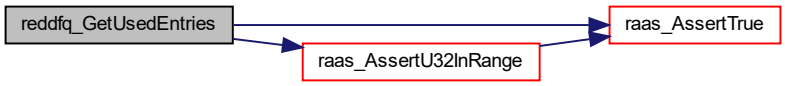
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

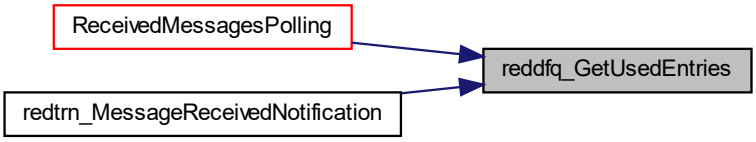
Returns

Number of used defer queue entries.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.2.6 reddfq_Init()

```
void reddfq_Init (
        const uint32_t configured_red_channels,
        const uint32_t configured_defer_queue_size,
        const uint32_t configured_t_seq )
```

Initialization of the data of the RedL defer queue module.

This function is used to initialize the defer queue module. It saves the passed number of redundancy channels, the defer queue size and defer queue timeout time Tseq. For all configured channels, the reddfq_InitDeferQueue() function is called to properly initialize the defer queues for all configured channels. A fatal error is raised, if this function is called multiple times.

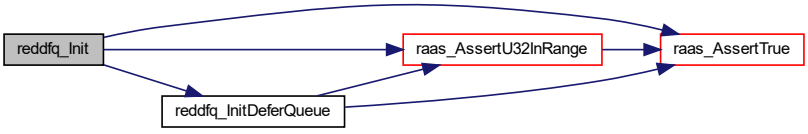
Precondition

The defer queue module must not be initialized, otherwise a radef_kAlreadyInitialized fatal error is thrown.

Parameters

in	*configured_red_channels*	Number of configured redundancy channels. Valid range: 1 <= value <= RADEF_MAX_NUMBER_OF_RED_CHANNELS.
in	*configured_defer_queue_size*	Configured defer queue size [messages]. Valid range: redcty_kMinDeferQueueSize <= value <= RADEF_MAX_DEFER_QUEUE_SIZE
in	*configured_t_seq*	Configured Tseq [ms]. Valid range: redcty_kMinTSeq <= value <= redcty_kMaxTSeq.

Here is the call graph for this function:



Here is the caller graph for this function:

4.7.2.7 reddfq_InitDeferQueue() `void reddfq_InitDeferQueue ( const uint32_t red_channel_id )`

Initialization of the defer queue of a dedicated redundancy channel.

This function initializes the defer queue of a given redundancy channel. It resets the number of used elements and sets the message length of all elements in the defer queue to 0, to indicate that these defer queue elements are not used.

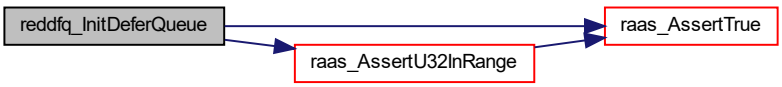
Precondition

The defer queue module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:

4.7.2.8 reddfq_IsSequenceNumberOlder()

```
bool reddfq_IsSequenceNumberOlder (
        const uint32_t sequence_number_to_compare,
        const uint32_t sequence_number_reference )
```

Returns true, if the sequence_number_to_compare is older than sequence_number_reference.

This function checks if sequence_number_to_compare is older than sequence_number_reference with respect to uint32_t wrap around for up counted sequence numbers. The limit to detect the older value is a difference of sequence_number_reference - (sequence_number_to_compare of + 1U) $<$ (UINT32_MAX / 2U).

Parameters

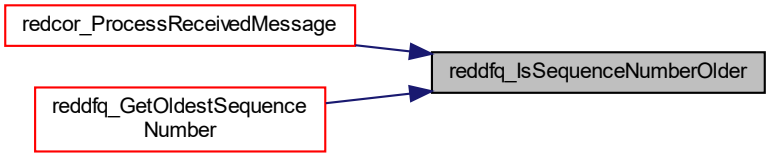
sequence_number_reference	Sequence number reference in comparison. The full value range is valid and usable.
sequence_number_to_compare	Sequence number to compare. The full value range is valid and usable.

Returns

true, if sequence_number_to_compare $<$ sequence_number_reference

false, if sequence_number_to_compare $>=$ sequence_number_reference

Here is the caller graph for this function:



4.7.2.9 reddfq_IsTimeout() `bool reddfq_IsTimeout ( const uint32_t red_channel_id )`

Check defer queue timeout on a dedicated redundancy channel.

This function checks if a message in the defer queue fulfils the defer queue timeout criteria: rasys_GetTimerValue() - message received timestamp > Tseq

Precondition

The defer queue module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

Parameters

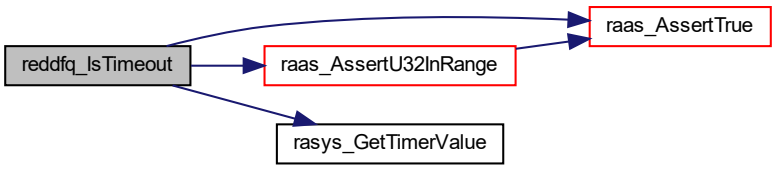
in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Returns

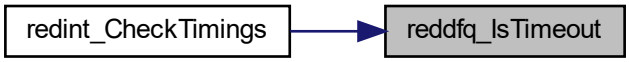
true, if there is a defer queue timeout.

false, if there is no defer queue timeout.

Here is the call graph for this function:



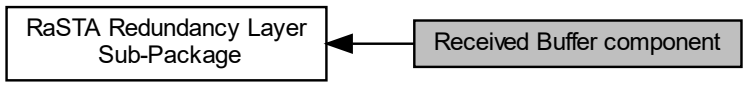
Here is the caller graph for this function:



4.8 Received Buffer component

Interface of RaSTA redundancy layer received messages buffer module.

Collaboration diagram for Received Buffer component:



Functions

- void redrbf_Init (const uint32_t configured_red_channels)

  Initialize the RedL received buffer module.
- void redrbf_InitBuffer (const uint32_t red_channel_id)

  Initialize the received buffer of a dedicated redundancy channel.
- void redrbf_AddToBuffer (const uint32_t red_channel_id, const redtyp_RedundancyMessagePayload *const message_payload)

  Add a RedL message to the received buffer of a dedicated redundancy channel. A fatal error is raised, if the buffer is full.
- radef_RaStaReturnCode redrbf_ReadFromBuffer (const uint32_t red_channel_id, const uint16_t buffer_size, uint16_t *const message_size, uint8_t *const message_buffer)

  Read and remove a RedL message payload from the received buffer of a dedicated redundancy channel.
- uint16_t redrbf_GetFreeBufferEntries (const uint32_t red_channel_id)

  Get the number of free buffer entries [messages].

4.8.1 Detailed Description

Interface of RaSTA redundancy layer received messages buffer module.

Specification of the Received Buffer component of the RaSTA Redundancy Layer.

This module buffers the payload of successfully received redundancy layer messages, for the read from safety and retransmission layer. The received buffer is organized as a FIFO ring buffer. One buffer entry holds a redtyp_RedundancyMessagePayload struct, which contains the payload of redundancy layer PDU message.

4.8.2 Function Documentation

4.8.2.1 redrbf_AddToBuffer()

```
void redrbf_AddToBuffer (
        const uint32_t red_channel_id,
        const redtyp_RedundancyMessagePayload *const message_payload )
```

Add a RedL message to the received buffer of a dedicated redundancy channel. A fatal error is raised, if the buffer is full.

When there is free space in the buffer, a RedL 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 index and buffer length are updated. The message_payload->payload_size must be in the range from RADEF_SR_LAYER_MESSAGE_HEADER_SIZE to RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidParameter fatal error message is thrown.

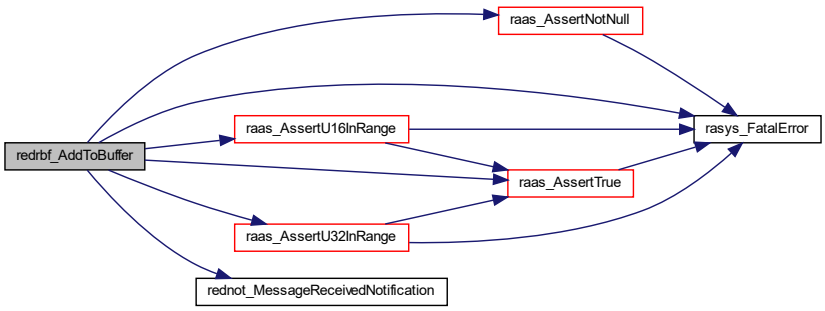
Precondition

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

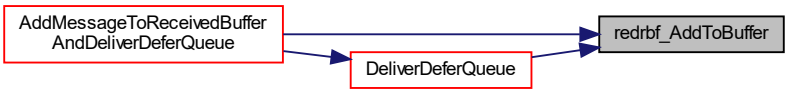
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*message_payload*	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.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.2 redrbf_GetFreeBufferEntries() `uint16_t redrbf_GetFreeBufferEntries ( const uint32_t red_channel_id )`

Get the number of free buffer entries [messages].

This function returns the amount of free entires in the received buffer of a given channel.

Precondition

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

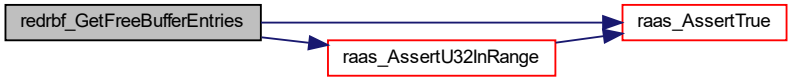
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

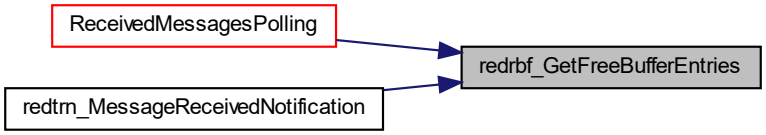
Returns

Number of free entries in the received buffer.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.3 redrbf_Init() `void redrbf_Init ( const uint32_t configured_red_channels )`

Initialize the RedL received buffer module.

This function is used to initialize the received buffer module. It saves the passed number of redundancy channels. For all configured channels, the redrbf_InitBuffer function is called to properly initialize the buffer for all configured channels. A fatal error is raised, if this function is called multiple times.

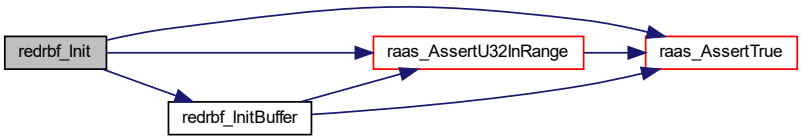
Precondition

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

Parameters

in	*configured_red_channels*	Number of configured redundancy channels. Valid range: 1 <= value <= RADEF_MAX_NUMBER_OF_RED_CHANNELS.

Here is the call graph for this function:



Here is the caller graph for this function:

4.8.2.4 redrbf_InitBuffer() `void redrbf_InitBuffer ( const uint32_t red_channel_id )`

Initialize the received buffer of a dedicated redundancy channel.

This function initializes the buffer of a given redundancy channel. It resets all properties of the buffer (read, write and used elements) and also sets the message length of all elements in the buffer to 0.

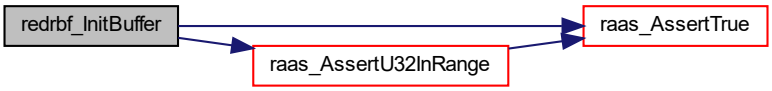
Precondition

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

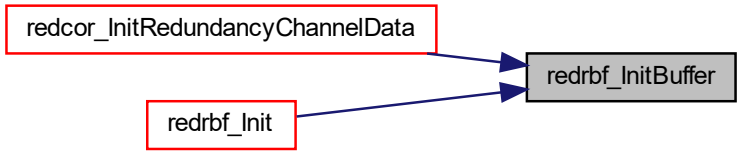
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.2.5 redrbf_ReadFromBuffer()

```
radef_RaStaReturnCode redrbf_ReadFromBuffer (
        const uint32_t red_channel_id,
        const uint16_t buffer_size,
        uint16_t *const message_size,
        uint8_t *const message_buffer )
```

Read and remove a RedL message payload from the received buffer of a dedicated redundancy channel.

When there are messages in the buffer, the oldest SafRetL message is read from the buffer, saved into the passed structure, the position index & length are updated and a radef_kNoError returned. If the buffer is empty, a radef_kNoMessageReceived is returned. The message payload size read from the buffer must be in the range from RADEF_SR_LAYER_MESSAGE_HEADER_SIZE to RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE, else a radef_kInternalError fatal error message is thrown.

Precondition

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

Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*buffer_size*	Size of the external memory buffer buffer to store the read message data [bytes]. The full value range is valid and usable.
out	*message_size*	Pointer to size of the read message data [bytes]. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
out	*message_buffer*	Pointer to external memory buffer used to store the read message data. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

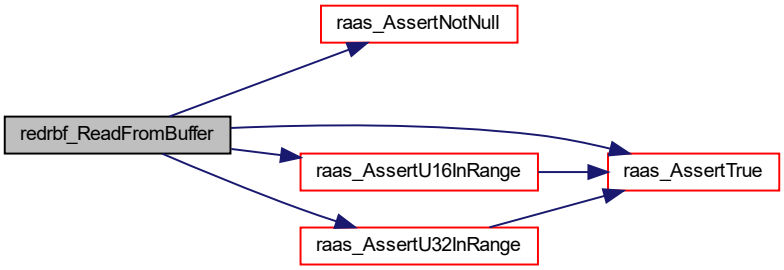
Returns

radef_kNoError -> Message successfully read from the buffer.

radef_kNoMessageReceived -> No received message in the buffer.

radef_kInvalidBufferSize -> External buffer size is too small for the current message.

Here is the call graph for this function:

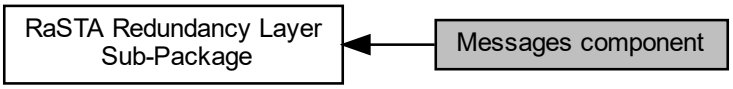


Here is the caller graph for this function:

4.9 Messages component

Interface of RaSTA redundancy layer messages module.

Collaboration diagram for Messages component:



Functions

- static void SetUint16InMessage (const uint16_t position, const uint16_t data, redtyp_RedundancyMessage *const red_message)

  Set a Uint16 at a specific position in a message.
- static void SetUint32InMessage (const uint16_t position, const uint32_t data, redtyp_RedundancyMessage *const red_message)

  Set a Uint32 at a specific position in a message.
- static void SetPayloadDataInMessage (const redtyp_RedundancyMessagePayload *const message_payload, redtyp_RedundancyMessage *const red_message)

  Set the payload data in a message.
- static uint16_t GetUint16FromMessage (const redtyp_RedundancyMessage *const red_message, const uint16_t position)

  Get a Uint16 from a specific position in a message.
- static uint32_t GetUint32FromMessage (const redtyp_RedundancyMessage *const red_message, const uint16_t position)

  Get a Uint32 from a specific position in a message.
- static uint16_t GetCheckCodeLength (const redcty_CheckCodeType check_code_type)

  Get the length of the configured check code.
- void redmsg_Init (const redcty_CheckCodeType configured_check_code_type)

  Initialize RedL messages module.
- void redmsg_CreateMessage (const uint32_t sequence_number, const redtyp_RedundancyMessagePayload *const message_payload, redtyp_RedundancyMessage *const redundancy_message)

  Create a new redundancy layer message and calculate the check code.
- radef_RaStaReturnCode redmsg_CheckMessageCrc (const redtyp_RedundancyMessage *const redundancy_message)

  Check the check code of a redundancy layer message.
- uint32_t redmsg_GetMessageSequenceNumber (const redtyp_RedundancyMessage *const redundancy_message)

  Get the sequence number of a redundancy layer message.
- void redmsg_GetMessagePayload (const redtyp_RedundancyMessage *const redundancy_message, redtyp_RedundancyMessagePayload *const message_payload)

  Get the payload of a redundancy message.

4.9.1 Detailed Description

Interface of RaSTA redundancy layer messages module.

Specification of the Messages component of the RaSTA Redundancy Layer.

This module provides all needed functionality for redundancy layer messages. This contains the following:

- validate a RedL message
- create a new RedL message
- extract information form a RedL message

4.9.2 Function Documentation

4.9.2.1 GetCheckCodeLength() `static uint16_t GetCheckCodeLength ( const redcty_CheckCodeType check_code_type ) [static]`

Get the length of the configured check code.

This internal function returns the byte length of the configured check code.

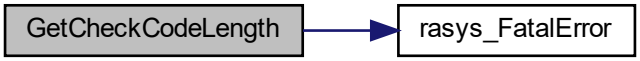
Parameters

in	*check_code_type*	check code type. All enum entries of redcty_CheckCodeType are valid and usable.

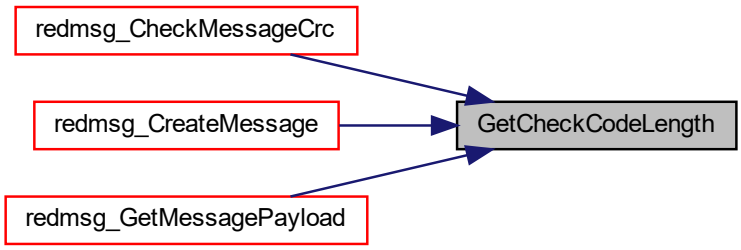
Returns

uint16_t length of the check 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 redtyp_RedundancyMessage *const red_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 > red↩ _message->message_size), a radef_kInternalError fatal error message is thrown. red_message->message_size must be set correctly before calling this function.

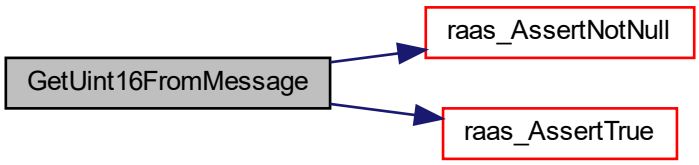
Parameters

in	*red_message*	Pointer to a message, from where a variable must be extracted. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	*position*	Start position where the variable must be read in the message [bytes].

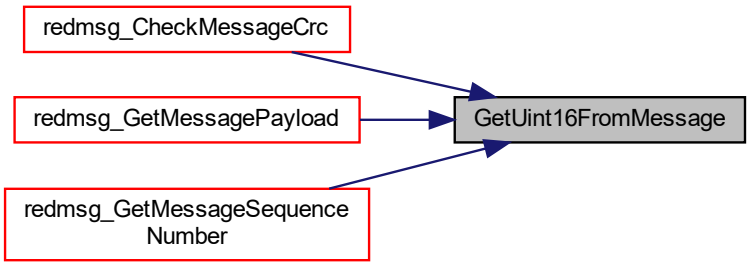
Returns

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 redtyp_RedundancyMessage *const red_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 + Uint32 byte size > red↩ _message->message_size), a radef_kInternalError fatal error message is thrown. red_message->message_size must be set correctly before calling this function.

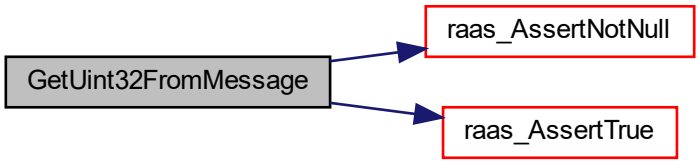
Parameters

in	*red_message*	Pointer to a message, from where a variable must be extracted. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in	*position*	Start position where the variable must be read in the message [bytes].

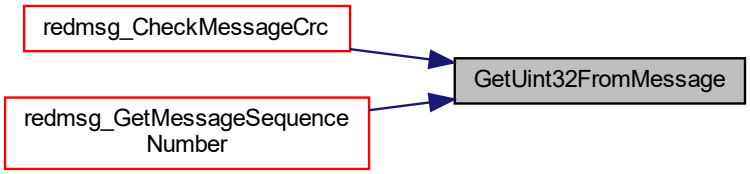
Returns

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 redmsg_CheckMessageCrc() `radef_RaStaReturnCode redmsg_CheckMessageCrc ( const redtyp_RedundancyMessage *const redundancy_message )`

Check the check code of a redundancy layer message.

This function checks the validity of a provided RedL message by checking the configured check code. For the check code type redcty_kCheckCodeA (no check code) it always returns radef_kNoError. The redundancy_message->message_size must be in the range from RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE to RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidParameter fatal error message is thrown. The redundancy_message->message_size must be equal to the message size stored in the message, else a radef_kInvalidParameter fatal error message is thrown.

Precondition

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

Parameters

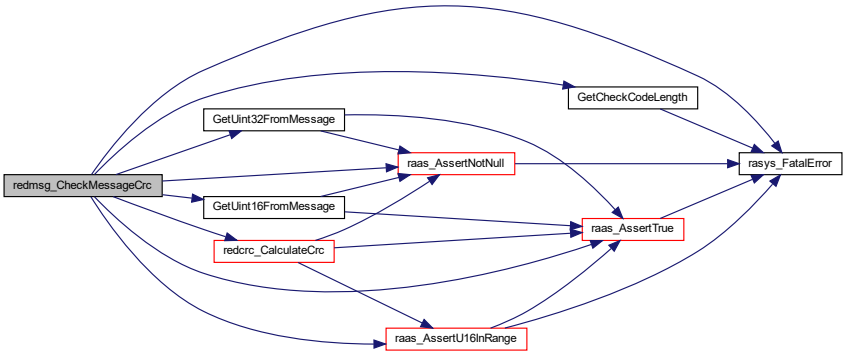
in	*redundancy_message*	pointer to message struct. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Returns

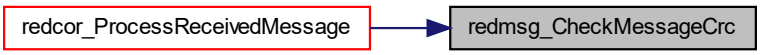
radef_kNoError -> check code is OK

radef_kInvalidMessageCrc -> wrong check code

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.5 redmsg_CreateMessage()

```
void redmsg_CreateMessage (
        const uint32_t sequence_number,
        const redtyp_RedundancyMessagePayload *const message_payload,
        redtyp_RedundancyMessage *const redundancy_message )
```

Create a new redundancy layer message and calculate the check code.

This function creates a new redundancy layer message:

- Calculate and set the message length according to the payload size and the check code type
- Initialize the reserve data bytes to 0

- Set the message sequence number
- Copy the message payload to the message
- Calculate and set the check code according to the check code type

The message_payload->payload_size must be in the range from RADEF_SR_LAYER_MESSAGE_HEADER_SIZE to RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidParameter fatal error message is thrown.

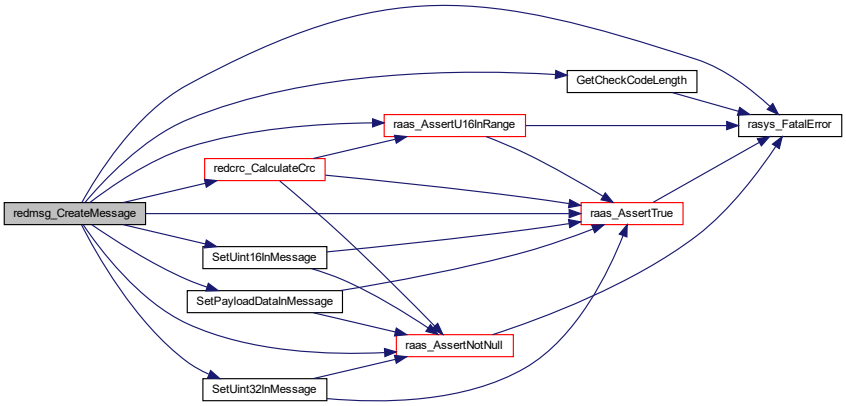
Precondition

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

Parameters

in	*sequence_number*	sequence number of the new message. The full value range is valid and usable.
in	*message_payload*	pinter to message payload data struct. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
out	*redundancy_message*	pointer to struct containing the new created 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:

4.9.2.6 redmsg_GetMessagePayload()

```
void redmsg_GetMessagePayload (
        const redtyp_RedundancyMessage *const redundancy_message,
        redtyp_RedundancyMessagePayload *const message_payload )
```

Get the payload of a redundancy message.

This function extracts the message payload from the passed RedL message and sets the message_payload->payload_size. The redundancy_message->message_size must be in the range from RADEF_MIN_RED_LAYER_PDU_MESSAGE_ to RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidParameter fatal error message is thrown. The redundancy_message->message_size must be equal to the message size stored in the message, else a radef_kInvalidParameter fatal error message is thrown. The internally calculated message payload size must be in the range from RADEF_SR_LAYER_MESSAGE_HEADER_SIZE to RADEF_MAX_SR_LAYER_PDU↩_MESSAGE_SIZE, else a radef_kInternalError fatal error message is thrown.

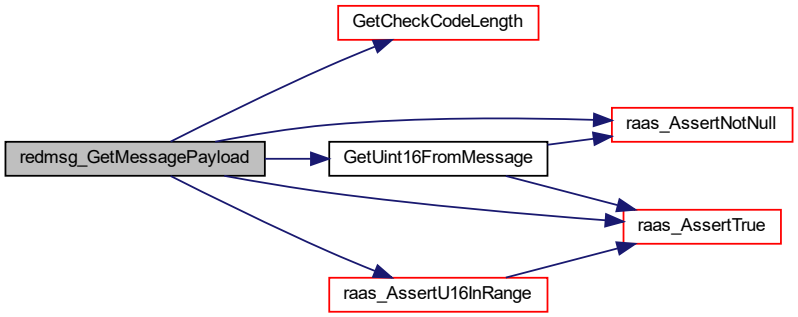
Precondition

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

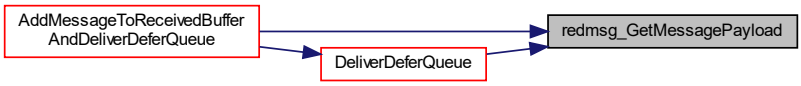
Parameters

in	*redundancy_message*	pointer to struct containing the message. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
out	*message_payload*	pointer to struct for the message payload. 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:



4.9.2.7 redmsg_GetMessageSequenceNumber() `uint32_t redmsg_GetMessageSequenceNumber ( const redtyp_RedundancyMessage *const redundancy_message )`

Get the sequence number of a redundancy layer message.

This function extracts the sequence number from the passed RedL message. The redundancy_↩ message->message_size must be in the range from RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE to RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE, else a radef_kInvalidParameter fatal error message is thrown. The redundancy_message->message_size must be equal to the message size stored in the message, else a radef_kInvalidParameter fatal error message is thrown.

Precondition

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

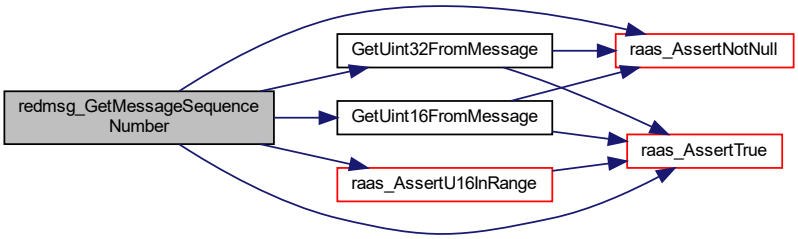
Parameters

in	*redundancy_message*	pointer to struct containing the message. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

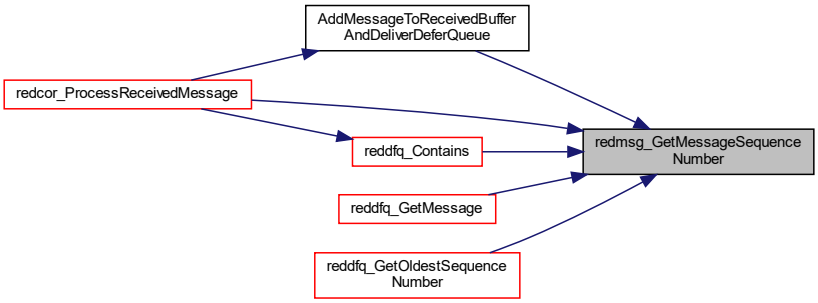
Returns

sequence number of the message

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.2.8 redmsg_Init() `void redmsg_Init (`

```
const redcty_CheckCodeType configured_check_code_type )
```

Initialize RedL messages module.

This function is used to initialize the messages module. It saves the passed check code type and calls the initialization of the CRC module. A fatal error is raised, if this function is called multiple times.

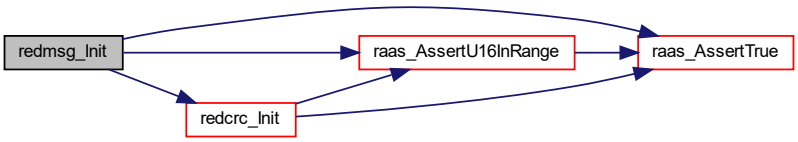
Precondition

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

Parameters

in	*configured_check_code_type*	Configured check code type. All enum entries of redcty_CheckCodeType are valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:

4.9.2.9 SetPayloadDataInMessage() `static void SetPayloadDataInMessage (`

```
const redtyp_RedundancyMessagePayload *const message_payload,
redtyp_RedundancyMessage *const red_message ) [static]
```

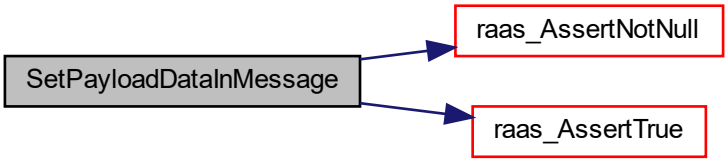
Set the payload data in a message.

This internal function writes specific payload data at a provided position in the message. If the payload doesn't fit inside the message (position + payload size > red_message->message_size), a radef_kInternalError fatal error message is thrown. red_message->message_size must be set correctly before calling this function.

Parameters

in	*message_payload*	Pointer to payload data that is written. If the pointer is NULL, a radef_kInternalError fatal error is thrown.
in,out	*red_message*	Pointer to a message, where the payload data 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:

4.9.2.10 SetUint16InMessage()

```
static void SetUint16InMessage (
        const uint16_t position,
        const uint16_t data,
        redtyp_RedundancyMessage *const red_message ) [static]
```

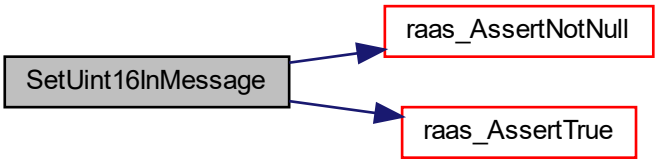
Set a Uint16 at a specific position in a message.

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 > red_message->message_size), a radef_kInternalError fatal error message is thrown. red_message->message_size must be set correctly before calling this function.

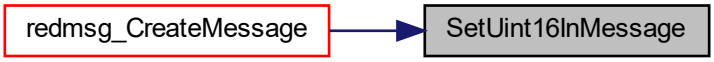
Parameters

in	*position*	Position where the variable must be set in the message [bytes].
in	*data*	Variable to be set in the provided message. The full value range is valid and usable.
in,out	*red_message*	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 SetUint32InMessage()

```
static void SetUint32InMessage (
        const uint16_t position,
        const uint32_t data,
        redtyp_RedundancyMessage *const red_message )  [static]
```

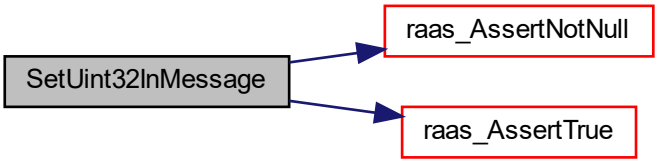
Set a Uint32 at a specific position in a message.

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 > red_message->message_size), a radef_kInternalError fatal error message is thrown. red_message->message_size must be set correctly before calling this function.

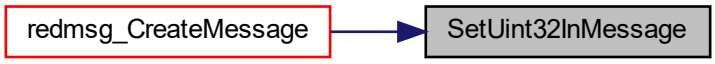
Parameters

in	*position*	Position where the variable must be set in the message [bytes].
in	*data*	Variable to be set in the provided message. The full value range is valid and usable.
in,out	*red_message*	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:



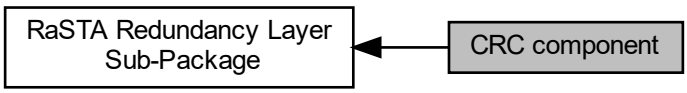
Here is the caller graph for this function:



4.10 CRC component

Interface of RaSTA redundancy layer CRC module.

Collaboration diagram for CRC component:



Functions

- static uint32_t ReflectBits (uint32_t value_in, uint16_t number_of_bits)

  Reflects the lower number_of_bits of a uint32_t and returns a value containing the reflected bits.
- static void GenerateCrcTable (void)

  Generates a CRC lookup table according to the configured redcrc_check_code_type.
- void redcrc_Init (const redcty_CheckCodeType configured_check_code_type)

  Initialize the CRC module and generate the CRC lookup table according to the configured check_code_type.
- void redcrc_CalculateCrc (const uint16_t data_size, const uint8_t ∗const data_buffer, uint32_t ∗const calculated_crc)

  Calculate the defined type of CRC of a data buffer.

4.10.1 Detailed Description

Interface of RaSTA redundancy layer CRC module.

Specification of the CRC component of the RaSTA Redundancy Layer.

This module provides all functionality needed to calculate the CRC values of RedL messages according to the following check code types defined for the RaSTA redundancy layer. All details can be found in the chapter 6.3.6 "Check code" of the standard "Elektrische Bahn-Signalanlagen – Teil 200: Sicheres Übertragungsprotokoll RaSTA nach DIN EN 50159 (VDE0831-159); Juni 2015".

Supported check code types:

Check code type A: No CRC check.

Check code type B:

- width = 32
- polynomial = 0xEE5B42FD
- initial_optimized = 0
- refin = false
- refout = false
- final_xor = 0

Check code type C:

- width = 32U
- polynomial = 0x1EDC6F41
- initial_optimized = 0xFFFFFFFF
- refin = true
- refout = true
- final_xor = 0xFFFFFFFF

Check code type D:

- width = 16
- polynomial = 0x1021
- initial_optimized = 0
- refin = true
- refout = true
- final_xor = 0

Check code type E:

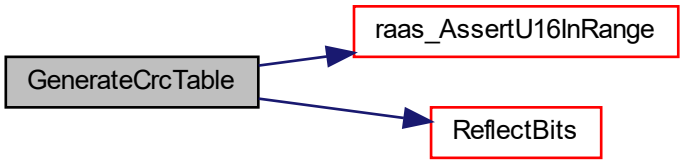
- width = 16
- polynomial = 0x8005
- initial_optimized = 0
- refin = true
- refout = true
- final_xor = 0

4.10.2 Function Documentation

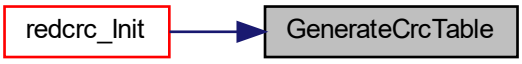
4.10.2.1 GenerateCrcTable() `static void GenerateCrcTable ( void ) [static]`

Generates a CRC lookup table according to the configured redcrc_check_code_type.

This function generates a CRC lookup table and sets the variables redcrc_crc_mask and redcrc_crc_high_bit according to the options in the configured redcrc_check_code_type. The generated CRC lookup table contains CRC_TABLE_SIZE elements. The configured check code type must be in the range: redcty_kCheckCodeB <= value < redcty_kCheckCodeMax. If the value is outside this range, a radef_kInternalError fatal error is thrown. The configured CRC width must be in the range: kMinWidth <= value <= kMaxWidth. 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:



4.10.2.2 redcrc_CalculateCrc() `void redcrc_CalculateCrc ( const uint16_t data_size, const uint8_t *const data_buffer, uint32_t *const calculated_crc )`

Calculate the defined type of CRC of a data buffer.

This function calculates the configured type of CRC value of the data_size bytes in the data_buffer. The configured check code type must be in the range: redcty_kCheckCodeB <= value < redcty_kCheckCodeMax. If the value is outside this range, a radef_kInternalError fatal error is thrown. The configured CRC width must be in the range: kMinWidth <= value <= kMaxWidth. If the value is outside this range, a radef_kInternalError fatal error is thrown.

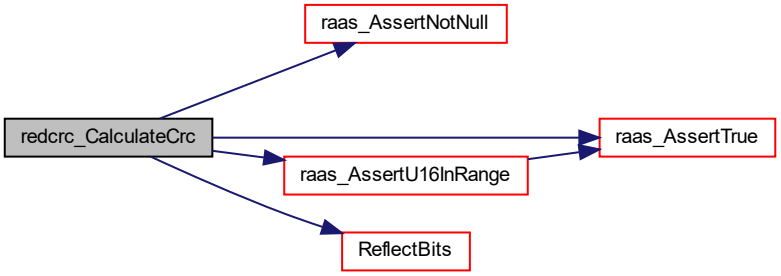
Precondition

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

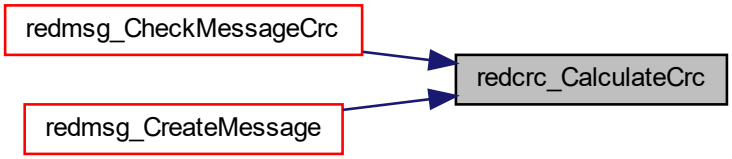
Parameters

in	*data_size*	Size of data buffer [bytes]. Valid range: RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE <= value <= RADEF_RED_LAYER_MESSAGE_HEADER_SIZE + RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE. If the value is outside this range, a radef_kInvalidParameter fatal error is thrown.
in	*data_buffer*	Pointer to data buffer containing the data for the CRC calculation. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.
out	*calculated_crc*	Pointer to calculated CRC. The full value range is valid and usable. 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:



4.10.2.3 redcrc_Init() `void redcrc_Init ( const redcty_CheckCodeType configured_check_code_type )`

Initialize the CRC module and generate the CRC lookup table according to the configured check_code_type.

This function is used to initialize the CRC module. It saves the passed check code type. A fatal error is raised, if this function is called multiple times. This function calls the internal function GenerateCrcTable(), which generates the CRC lookup table according to the configured check_code_type. For check code type A (no CRC check) no CRC lookup table is generated.

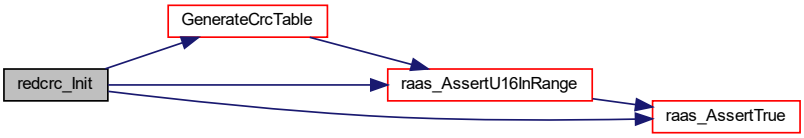
Precondition

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

Parameters

in	*configured_check_code_type*	Configured type of CRC check code. All enum entries of redcty_CheckCodeType are valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:

4.10.2.4 ReflectBits() `static uint32_t ReflectBits ( uint32_t value_in, uint16_t number_of_bits ) [static]`

Reflects the lower number_of_bits of a uint32_t and returns a value containing the reflected bits.

This function reflects the specified number of lower bits of the given value and returns a value containing the reflected bits.

Parameters

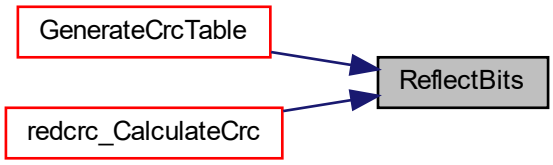
value_in	The input value to reflect. The full value range is valid and usable.
number_of_bits	The number of bits which will be reflected. Valid range: kMinWidth <= value <= kMaxWidth. If the value is outside this range, a radef_kInternalError fatal error is thrown.

Returns

The reflected output value. The full value range is valid and usable.

Here is the call graph for this function:

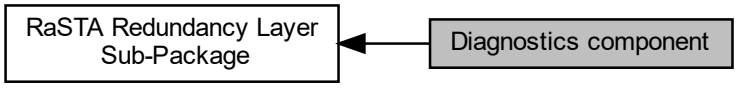
Here is the caller graph for this function:



4.11 Diagnostics component

Interface of RaSTA redundancy layer diagnostics.

Collaboration diagram for Diagnostics component:



Classes

- struct reddia_ReceivedMessageTimestamp

  Struct for the timestamps of first received messages.

Functions

- uint32_t GetTransportChannelIndex (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Get the transport channel index of a specific transport channel of a redundancy channel.
- static bool IsSequenceNumberAlreadyReceivedUpdateDiagnosticData (const uint32_t red_channel_id, const uint32_t transport_channel_id, const uint32_t message_sequence_number, const uint32_t current_time_↩ stamp)

  Returns true, if a received message timestamp of a message with the given sequence number is already stored. If so, it updates the diagnostic data of the transport channel of the newly received message.
- static void AddFirstTimeReceivedMessageDiagnosticData (const uint32_t red_channel_id, const uint32_↩ t transport_channel_id, const uint32_t message_sequence_number, const uint32_t current_time_stamp)

  Add the diagnostic data of a message with a first time received sequence number and trigger the diagnostic notifications, if the diagnostic window is reached.
- void reddia_InitRedundancyLayerDiagnostics (const redcty_RedundancyLayerConfiguration ∗const redundancy_layer_configuration)

  Initialize the RedL diagnostics module.
- void reddia_InitRedundancyChannelDiagnostics (const uint32_t red_channel_id)

  Initialize diagnostic data of a dedicated redundancy channel.
- void reddia_UpdateRedundancyChannelDiagnostics (const uint32_t red_channel_id, const uint32_↩ t transport_channel_id, const uint32_t message_sequence_number)

  Update redundancy channel diagnostic data with the data of a newly received message.
- bool reddia_IsConfigurationValid (const redcty_RedundancyLayerConfiguration ∗const redundancy_layer_configuration)

  Checks if the redundancy layer configuration is valid.
- bool reddia_IsTransportChannelIdValid (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Checks, if a transport channel identification is valid for a given redundancy channel.

4.11.1 Detailed Description

Interface of RaSTA redundancy layer diagnostics.

Specification of the Diagnostics component of the RaSTA Redundancy Layer.

This module provides the RaSTA RedL diagnostics functionality:

- Initialize the diagnostic data of a redundancy channel
- Update diagnostic data of a redundancy channel with the data of a newly received message
- Trigger diagnostic notifications with diagnostic data of the transport channels, if the diagnostic window for a redundancy channel is reached

Additional the following helper functions are provided:

- Check the validity of the RedL configuration
- Check the association of a transport channel to a redundancy channel

4.11.2 Function Documentation

4.11.2.1 AddFirstTimeReceivedMessageDiagnosticData()

```
static void AddFirstTimeReceivedMessageDiagnosticData (
            const uint32_t red_channel_id,
            const uint32_t transport_channel_id,
            const uint32_t message_sequence_number,
            const uint32_t current_time_stamp ) [static]
```

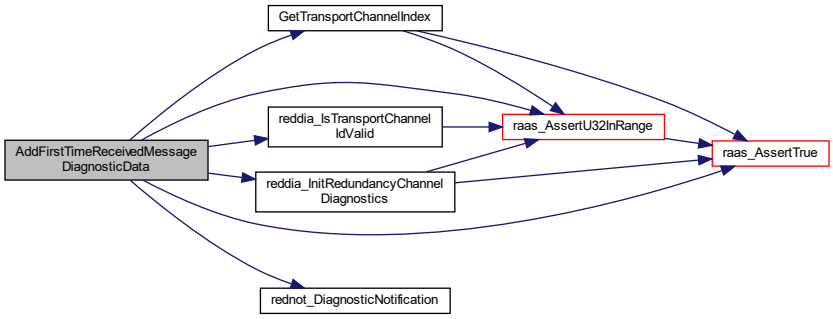
Add the diagnostic data of a message with a first time received sequence number and trigger the diagnostic notifications, if the diagnostic window is reached.

This Function checks, if the configured diagnosis window is reached for the given redundancy channel. If so, the diagnostic data is collected and the diagnostic notifications are triggered for all associated transport channels by calling rednot_DiagnosticNotification() and subsequently the diagnostic data of this redundancy channel is reset by calling reddia_InitRedundancyChannelDiagnostics(). Finally the timestamp, sequence number, transport channel id are stored in reddia_received_messages_timestamps and the reddia_current_n_diagnosis of this transport channel is increased. If reddia_current_n_diagnosis of the given redundancy channel is greater than n_diagnosis a radef_kInternalError fatal error is thrown.

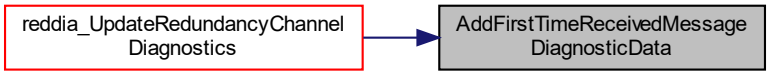
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the specified redundancy channel, a radef_kInternalError fatal error is thrown.
in	*message_sequence_number*	Newly received message sequence number. The full value range is valid and usable.
	current_time_stamp	Current timestamp [ms]. The full value range is valid and usable.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.2 GetTransportChannelIndex() `uint32_t GetTransportChannelIndex ( const uint32_t red_channel_id, const uint32_t transport_channel_id )`

Get the transport channel index of a specific transport channel of a redundancy channel.

The function returns the index of the transport channel in the redundancy channel configuration structure of a specific redundancy channel.

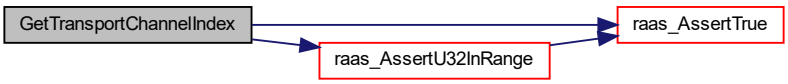
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the specified redundancy channel, a radef_kInternalError fatal error is thrown.

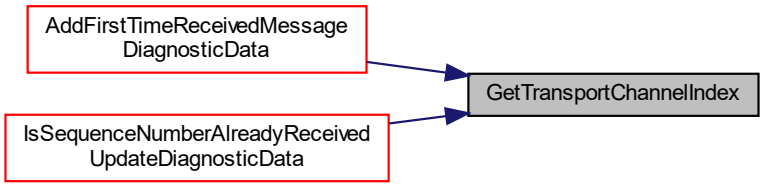
Returns

The index of the transport channel in the redundancy channel configuration structure of a specific redundancy channel.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.3 IsSequenceNumberAlreadyReceivedUpdateDiagnosticData()

```
static bool IsSequenceNumberAlreadyReceivedUpdateDiagnosticData (
            const uint32_t red_channel_id,
            const uint32_t transport_channel_id,
            const uint32_t message_sequence_number,
            const uint32_t current_time_stamp ) [static]
```

Returns true, if a received message timestamp of a message with the given sequence number is already stored. If so, it updates the diagnostic data of the transport channel of the newly received message.

First, this function checks if a received message timestamp of a message with the given sequence number is already stored in reddia_received_messages_timestamps. If so, radef_TransportChannelDiagnosticData.t_drift, radef_TransportChannelDiagnosticData.t_drift2 and the reddia_ReceivedMessageTimestamp.message_received_flag of the given transport channel are updated.

Precondition

The reddia_current_n_diagnosis of the given redundancy channel must be smaller or equal to the configured n_diagnosis, else a radef_kInternalError fatal error is thrown.

Parameters

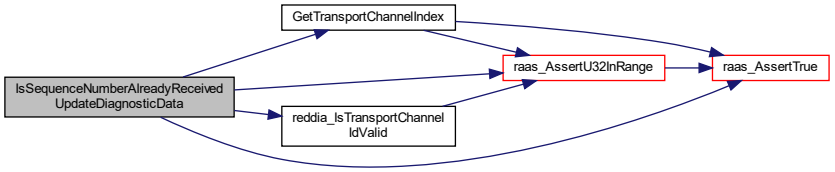
in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the specified redundancy channel, a radef_kInternalError fatal error is thrown.
in	*message_sequence_number*	Newly received message sequence number. The full value range is valid and usable.
	current_time_stamp	Current timestamp [ms]. The full value range is valid and usable.

Returns

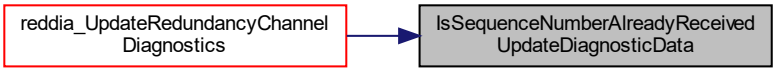
true, if a message with the same sequence number was already received before

false, if no message with the given sequence number was received before

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.4 reddia_InitRedundancyChannelDiagnostics() `void reddia_InitRedundancyChannelDiagnostics (`

```
            const uint32_t red_channel_id )
```

Initialize diagnostic data of a dedicated redundancy channel.

This function initializes the diagnostic data of a given redundancy channel. It resets the following properties:

- timestamps include complete reddia_ReceivedMessageTimestamp structure
- Ndiagnosis
- Nmissed
- Tdrift
- Tdrift^2

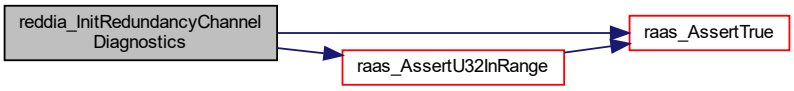
Precondition

The RedL diagnostics module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

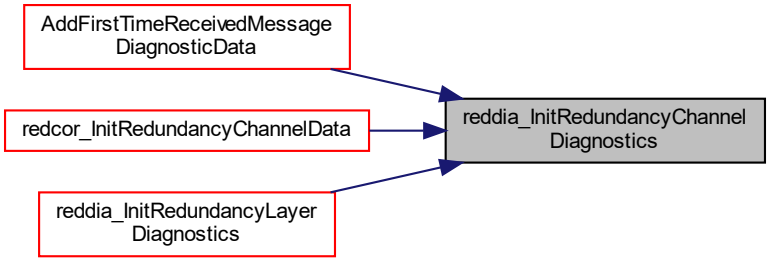
Parameters

in	*red_channel↩ _id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.5 reddia_InitRedundancyLayerDiagnostics() `void reddia_InitRedundancyLayerDiagnostics ( const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration )`

Initialize the RedL diagnostics module.

This function is used to initialize the RedL diagnostics module. The validity of the configuration is checked by calling the reddia_IsConfigurationValid() function. It saves the passed redundancy layer configuration pointer. For all configured channels, the reddia_InitRedundancyChannelDiagnostics() function is called to properly initialize the diagnostic data for all configured channels.

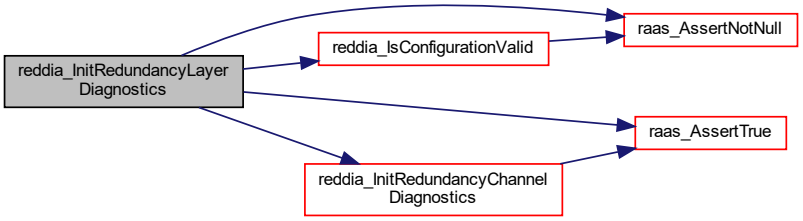
Precondition

The RedL diagnostics module must not be initialized, otherwise a radef_kAlreadyInitialized fatal error is thrown.

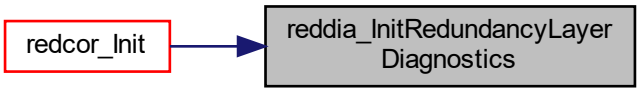
Parameters

in	*redundancy_layer_configuration*	Pointer to redundancy layer configuration data structure. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown. If the configuration is not valid a::radef_kInvalidConfiguration fatal error is thrown.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.6 reddia_IsConfigurationValid() `bool reddia_IsConfigurationValid ( const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration )`

Checks if the redundancy layer configuration is valid.

This function checks if all elements of the redundancy layer configuration are in their valid ranges and if all configuration elements are consistent to the others. A configuration is valid if all of the following conditions are met:

- redcty_kCheckCodeMin <= check_code_type < redcty_kCheckCodeMax
- redcty_kMinTSeq <= t_seq <= redcty_kMaxTSeq
- redcty_kMinRedLayerNDiagnosis <= n_diagnosis <= RADEF_MAX_RED_LAYER_N_DIAGNOSIS
- redcty_kMinDeferQueueSize <= n_defer_queue_size <= RADEF_MAX_DEFER_QUEUE_SIZE
- redcty_kMinNumberOfRedundancyChannels <= number_of_redundancy_channels <= RADEF_MAX_NUMBER_OF_RED_C
- red_channel_ids are identical to their index in the configuration data structure
- redcty_kMinNumberOfTransportChannels <= num_transport_channels <= RADEF_MAX_NUMBER_OF_TRANSPORT_CHA
- 0 <= transport_channel_ids < (RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPOR

Parameters

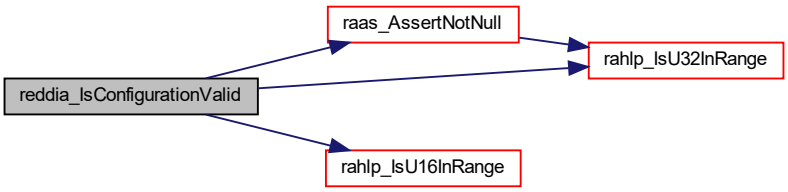
in	*redundancy_layer_configuration*	Pointer to redundancy layer configuration data structure. If the pointer is NULL, a radef_kInvalidParameter fatal error is thrown.

Returns

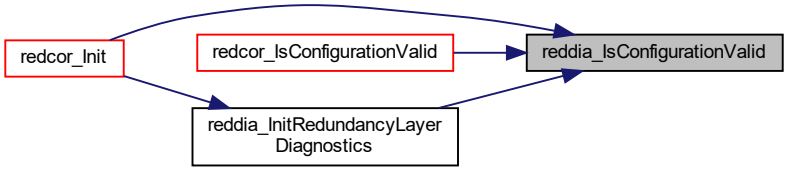
true, if the configuration is valid.

false, if the configuration is invalid.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.7 reddia_IsTransportChannelIdValid()

```
bool reddia_IsTransportChannelIdValid (
        const uint32_t red_channel_id,
        const uint32_t transport_channel_id )
```

Checks, if a transport channel identification is valid for a given redundancy channel.

This function checks, if a transport channel identification is found in the configuration of a given redundancy channel.

Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
	transport_channel↩ _id	Transport channel identification to check. The full value range is valid, but if the transport channel is not associated to the redundancy channel, the function

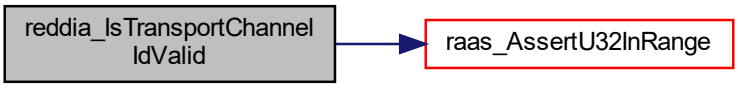
returns false.

Returns

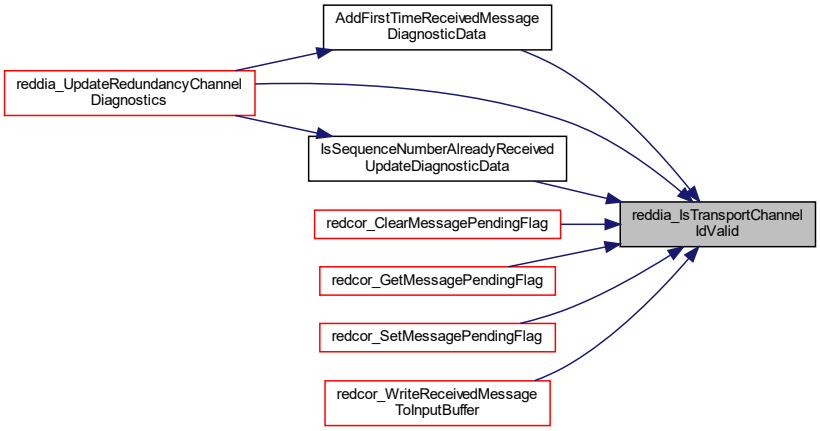
true, if the transport channel identification is found in the configuration of the given redundancy channel

false, if the transport channel identification is not found in the configuration of the given redundancy channel

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.2.8 reddia_UpdateRedundancyChannelDiagnostics()

```
void reddia_UpdateRedundancyChannel↩
Diagnostics (
            const uint32_t red_channel_id,
            const uint32_t transport_channel_id,
            const uint32_t message_sequence_number )
```

Update redundancy channel diagnostic data with the data of a newly received message.

This function updates the redundancy channel diagnostic data with the data of a newly received message. If a message with a specific sequence number is received for the first time, the sequence number and a message received timestamp of this message is saved and the internal message received flag for this transport channel is set and the current Ndiagnosis (message counter) of the given redundancy channel is increased. If the sequence number of a newly received message is found in the already received message sequence numbers, the difference to the first message received timestamp to the current time is calculated. If this difference is smaller than the

configured Tseq, the difference is added to Tdrift and the square of the difference is added to Tdrift^2 and the internal message received flag of the transport channel is set. If the configured RedL diagnosis window is reached for this redundancy channel, the Nmissed of each transport channel is calculated out of the internal message received flags and a diagnostic notification is sent to all associated transport channels and the diagnostic data of this redundancy channel is cleared by calling reddia_InitRedundancyChannelDiagnostics(). If the current Ndiagnosis is greater than the configured RedL diagnosis window size, a radef_kInternalError fatal error is thrown.

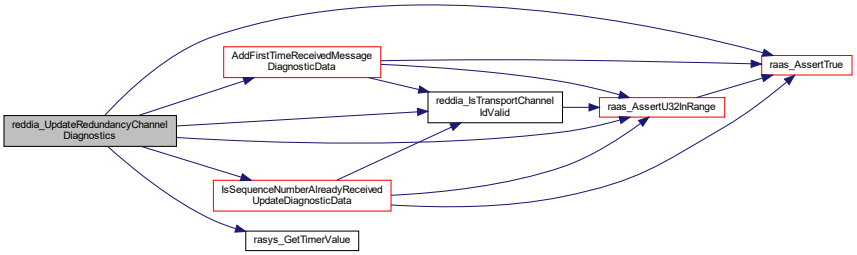
Precondition

The RedL diagnostics module must be initialized, otherwise a radef_kNotInitialized fatal error is thrown.

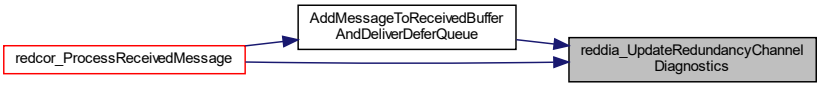
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.
in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of the specified redundancy channel, a radef_kInvalidParameter fatal error is thrown.
in	*message_sequence_number*	Newly received message sequence number. The full value range is valid and usable.

Here is the call graph for this function:



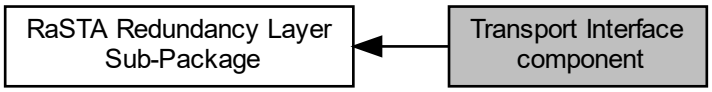
Here is the caller graph for this function:



4.12 Transport Interface component

Interface of RaSTA transport layer.

Collaboration diagram for Transport Interface component:



Functions

- void redtri_Init (void)

  Initialize transport layer.
- void redtri_SendMessage (const uint32_t transport_channel_id, const uint16_t message_size, const uint8_t ∗const message_data)

  Send a RedL message over a transport channel.
- radef_RaStaReturnCode redtri_ReadMessage (const uint32_t transport_channel_id, const uint16_t buffer↩ _size, uint16_t ∗const message_size, uint8_t ∗const message_buffer)

  Read a received RedL message from a transport channel.

4.12.1 Detailed Description

Interface of RaSTA transport layer.

Specification of the Transport Interface component of the RaSTA Redundancy Layer.

This module defines the interface functions (like init, send & read message) for the transport layer interface. The RedL only defines the interface, the implementation of this interface functions must be done in the transport layer.

Remarks

The error handling for all function must be implemented and handled by the system integrator when developing the transport layer.

4.12.2 Function Documentation

4.12.2.1 redtri_Init() `void redtri_Init (`

```
void )
```

Initialize transport layer.

This function is used to initialize the transport layer. The RedL only defines the interface function, the implementation of this interface function must be done in the transport layer.

4.12.2.2 redtri_ReadMessage() `radef_RaStaReturnCode redtri_ReadMessage (`

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

Read a received RedL message from a transport channel.

This function is used to read a RedL message from a specific transport channel of the transport layer. The RedL only defines the interface function, the implementation of this interface function must be done in the transport layer.

Parameters

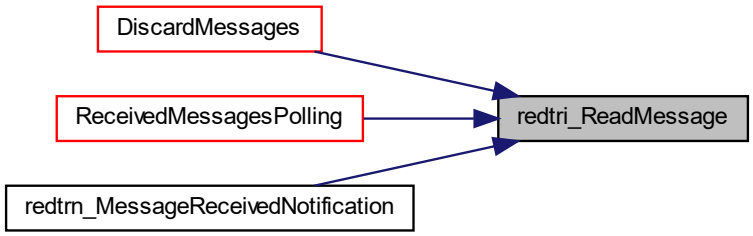
in	*transport_channel↩ _id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS.
in	*buffer_size*	Size of the buffer [bytes]. Valid range: RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE <= value <= RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE.
out	*message_size*	Pointer to the size of the received message data [bytes].
out	*message_buffer*	Pointer to a buffer for saving the received message. Enough memory to save a message with buffer_size must be allocated.

Returns

radef_kNoError -> successful operation

radef_kNoMessageReceived -> no message received (used for polling)

Here is the caller graph for this function:



4.12.2.3 redtri_SendMessage()

```
void redtri_SendMessage (
    const uint32_t transport_channel_id,
    const uint16_t message_size,
    const uint8_t *const message_data )
```

Send a RedL message over a transport channel.

This function is used to send a RedL message over a specific transport channel of the transport layer. The RedL only defines the interface function, the implementation of this interface function must be done in the transport layer.

Parameters

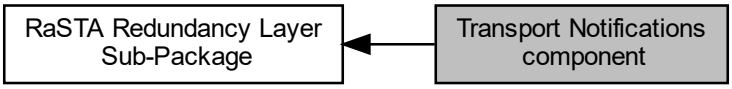
in	*transport_channel↩ _id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS.
in	*message_size*	Size of the message data [bytes]. Valid range: RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE <= value <= RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE.
in	*message_data*	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 Transport Notifications component

Interface of RaSTA transport layer notifications.

Collaboration diagram for Transport Notifications component:



Functions

- void redtrn_MessageReceivedNotification (const uint32_t transport_channel_id)

  Transport layer message received notification function.

4.13.1 Detailed Description

Interface of RaSTA transport layer notifications.

Specification of the Transport Notifications component of the RaSTA Redundancy Layer.

This module defines and implements the transport layer message received notification, as it acts as entry point to the RedL for the transport layer.

4.13.2 Function Documentation

4.13.2.1 redtrn_MessageReceivedNotification() `void redtrn_MessageReceivedNotification ( const uint32_t transport_channel_id )`

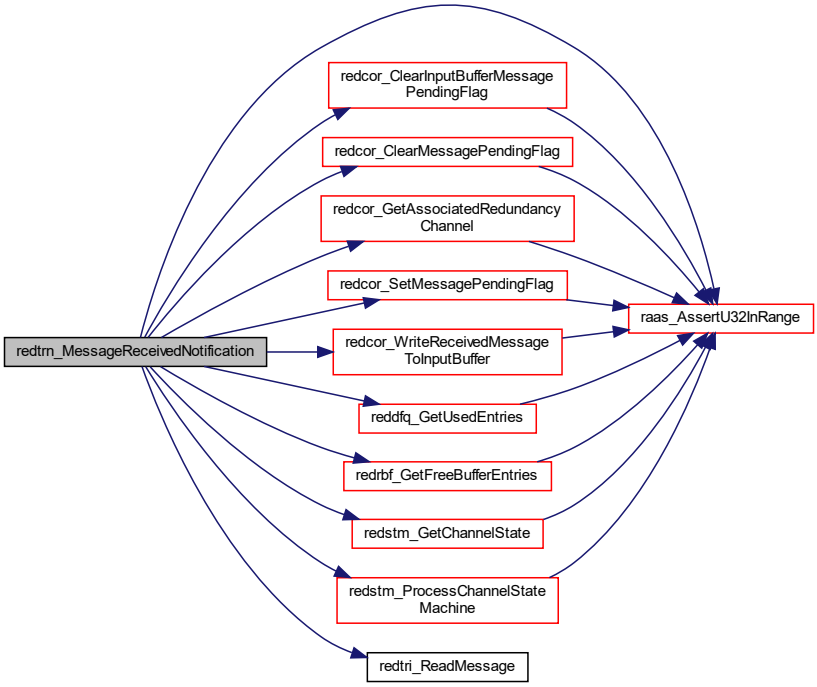
Transport layer message received notification function.

This function is called by the transport layer to notify the RedL that a new received message is ready to be read. If the associated redundancy channel state machine is in the up state and there are more free received buffer entries than used defer queue entries, the message is read from the transport channel, copied to the input buffer and the state machine is triggered with a redstm_kRedundancyChannelEventReceiveData event, to process the received message. If there are not enough free buffer entries in the received buffer, the message is not read and the received message pending flag of this transport channel is set. If the associated redundancy channel state machine is not in the up state, the message is read and discarded. If in any case there was no message to read, the received message pending flag of this transport channel is cleared.

Parameters

in	*transport_channel_id*	Transport channel identification. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_RED_CHANNELS * RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS. If the transport channel id is out of range, or the transport channel id is not in the configuration of a redundancy channel, a radef_kInvalidParameter fatal error is thrown.

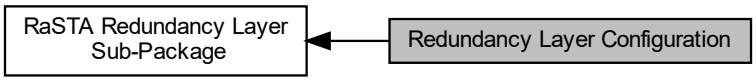
Here is the call graph for this function:



4.14 Redundancy Layer Configuration

Type definitions of RaSTA redundancy layer configuration.

Collaboration diagram for Redundancy Layer Configuration:



Classes

- struct redcty_RedundancyChannelConfiguration
  Struct for the configuration data of a redundancy channel.
- struct redcty_RedundancyLayerConfiguration
  Struct for the configuration data of the redundancy layer.

Enumerations

- enum redcty_CheckCodeType {
  redcty_kCheckCodeMin = 0 , redcty_kCheckCodeA = 0 , redcty_kCheckCodeB = 1 , redcty_kCheckCodeC = 2 ,
  redcty_kCheckCodeD = 3 , redcty_kCheckCodeE = 4 , redcty_kCheckCodeMax }
  Enum for the check code type of the redundancy channels.

Variables

- const uint32_t **redcty_kMinNumberOfRedundancyChannels**
  Minimum number of redundancy channels.
- const uint32_t **redcty_kMinNumberOfTransportChannels**
  Minimum number of transport channels.
- const uint32_t **redcty_kMinTSeq**
  Minimum time for out of sequence message buffering (Tseq) [ms].
- const uint32_t **redcty_kMaxTSeq**
  Maximum time for out of sequence message buffering (Tseq) [ms].
- const uint32_t **redcty_kMinRedLayerNDiagnosis**
  Minimum diagnosis window size [messages].
- const uint32_t **redcty_kMinDeferQueueSize**
  Minimum size of a redundancy channel defer queue [messages].

4.14.1 Detailed Description

Type definitions of RaSTA redundancy layer configuration.

Specification of the Configuration of the RaSTA Redundancy Layer.

This module defines the data types and data structures used for the RaSTA RedL configuration.

4.14.2 Enumeration Type Documentation

4.14.2.1 redcty_CheckCodeType `enum redcty_CheckCodeType`

Enum for the check code type of the redundancy channels.

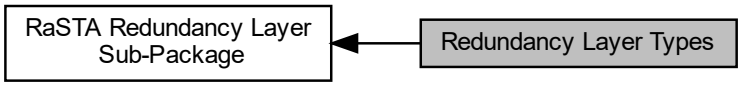
Enumerator

redcty_kCheckCodeMin	Min value for check code enum.
redcty_kCheckCodeA	Check code type a) is used: No check code.
redcty_kCheckCodeB	Check code type b) is used: CRC32 with polynomial 0xEE5B42FD.
redcty_kCheckCodeC	Check code type c) is used: CRC32 with polynomial 0x1EDC6F41.
redcty_kCheckCodeD	Check code type d) is used: CRC16 with polynomial 0x1021.
redcty_kCheckCodeE	Check code type e) is used: CRC16 with polynomial 0x8005.
redcty_kCheckCodeMax	Max value for check code enum.

4.15 Redundancy Layer Types

Internal type definitions of RaSTA redundancy layer.

Collaboration diagram for Redundancy Layer Types:



Classes

- struct redtyp_RedundancyMessage

  Typedef for a redundancy layer PDU message.
- struct redtyp_RedundancyMessagePayload

  Typedef for a redundancy layer PDU message payload.

4.15.1 Detailed Description

Internal type definitions of RaSTA redundancy layer.

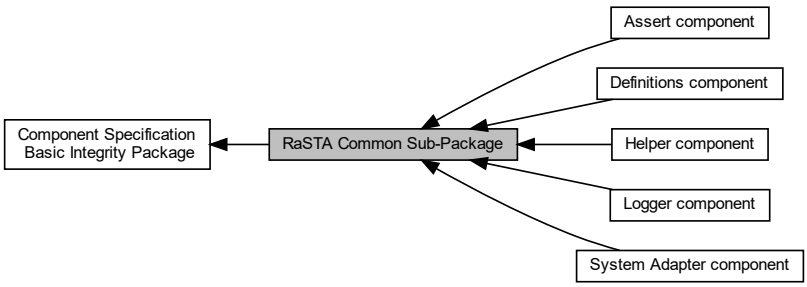
Specification of the internal type structures of the RaSTA Redundancy Layer.

Changelog **-:** Initial version (-, -)

This module defines the internal data types of the RaSTA RedL like the RedL PDU message and the RedL PDU message payload.

4.16 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.

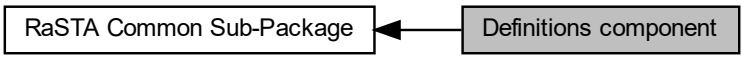
4.16.1 Detailed Description

Specification of all components within the RaSTA Common Package.

4.17 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_I - 1U)

  Size of timing distribution diagnostic interval array. Contains one element less then intervals since last element is set to t_max.
- #define **RADEF_MAX_N_SEND_MAX** (20U)

  Maximum number of entires 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.

4.17.1 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

4.17.2 Macro Definition Documentation

4.17.2.1 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].

4.17.2.2 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".

4.17.3 Enumeration Type Documentation

4.17.3.1 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

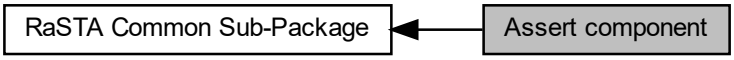
Enumerator

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.

4.18 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.

4.18.1 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

4.18.2 Function Documentation

4.18.2.1 raas_AssertNotNull()

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

Assert if a pointer is not NULL.

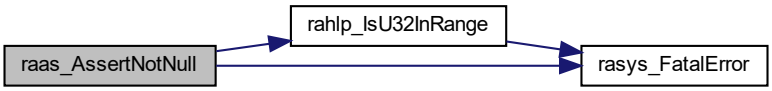
This function checks if a pointer is not NULL. If that is not the case, the passed error code is thrown as fatal error using rasys_FatalError.

Implements Requirements RASW-534 Assert not Null Function
RASW-521 Input Parameter Check

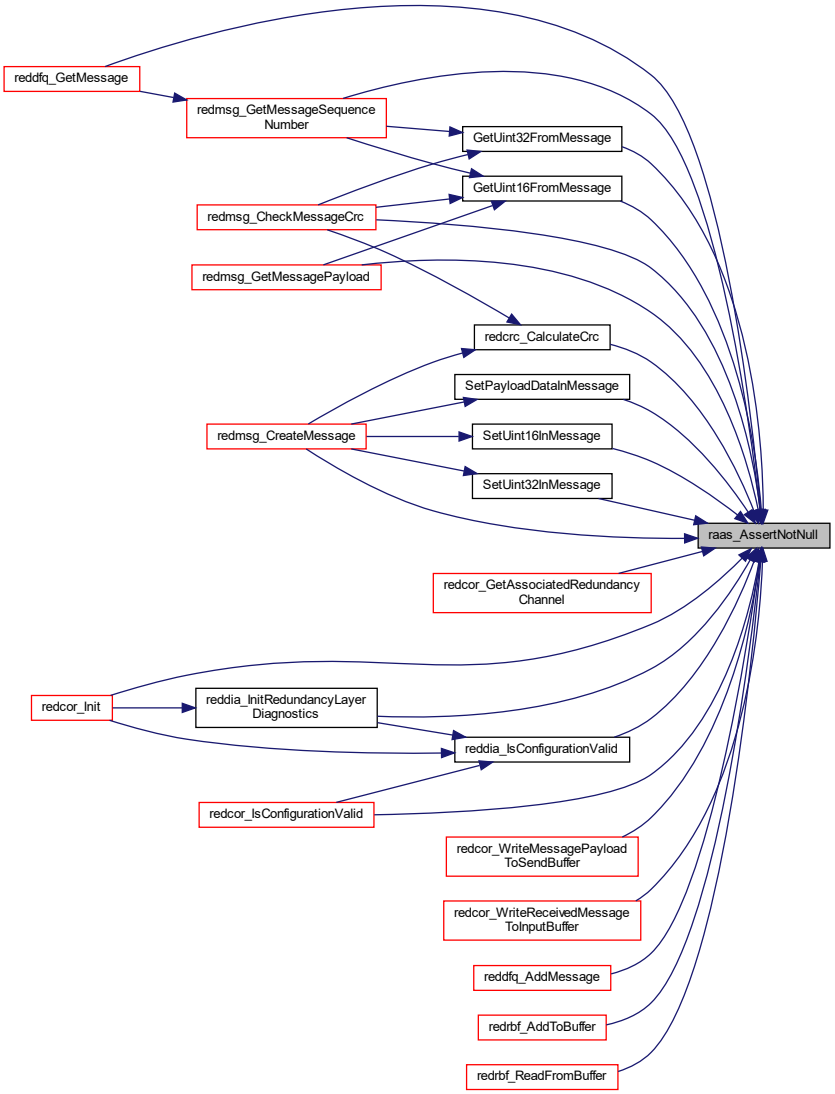
Parameters

in	*pointer*	Pointer to check.
in	*error_reason*	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:



4.18.2.2 raas_AssertTrue()

```
void raas_AssertTrue (
        const bool condition,
        const radef_RaStaReturnCode error_reason )
```

Assert a bool condition is true.

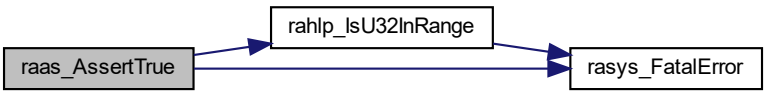
This function checks if a bool condition is true. If that is not the case, the passed error code is thrown as fatal error using rasys_FatalError.

Implements Requirements RASW-535 Assert True Function
RASW-521 Input Parameter Check

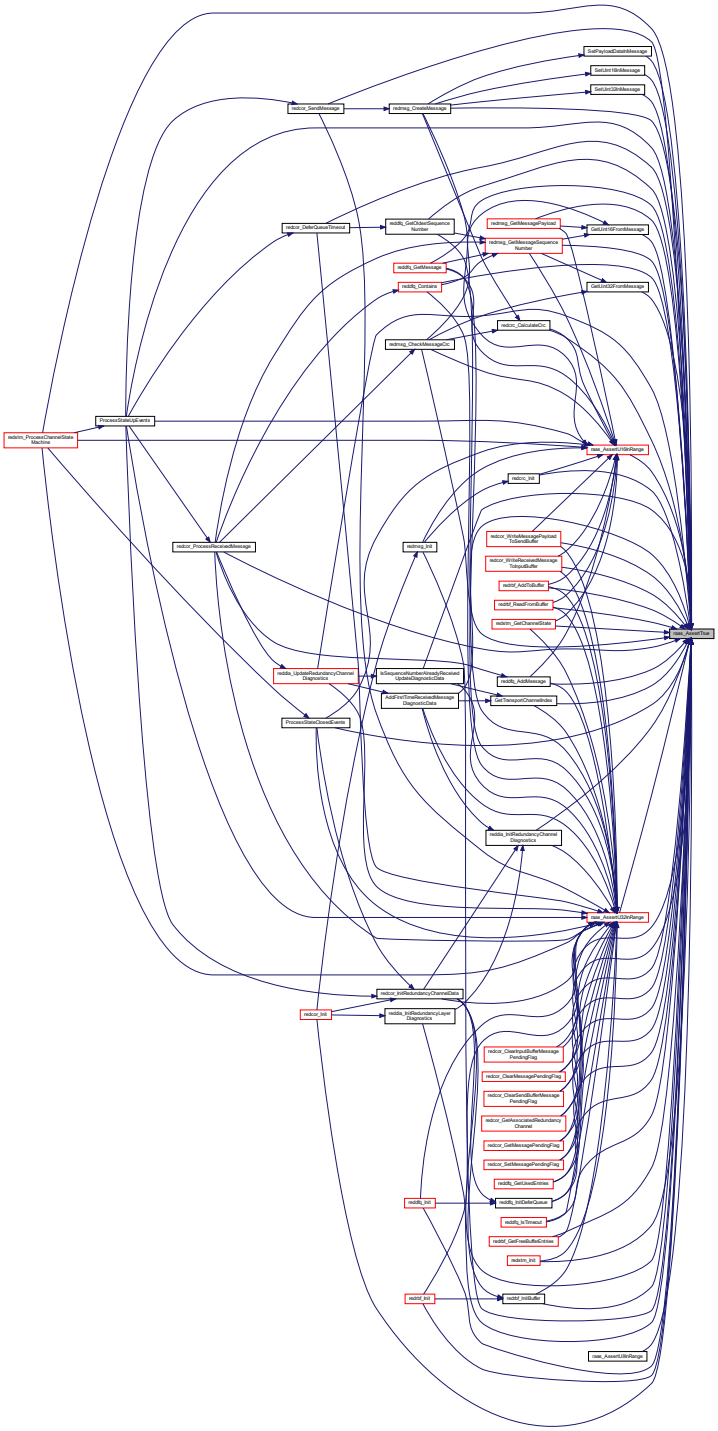
Parameters

in	*condition*	Condition to check.
in	*error_reason*	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:



4.18.2.3 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.

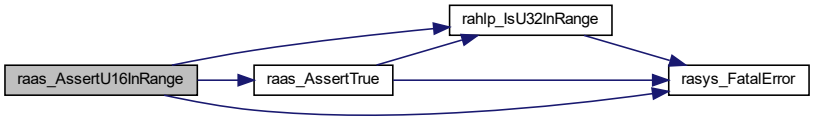
This function checks if a uint16_t value is inside a defined range. Check condition: min_value <= value <= max↩ _value. The minimum value must be smaller or equal to the maximum value, otherwise a radef_kInvalidParameter fatal error is thrown.

Implements Requirements RASW-536 Assert U16 in Range Function
RASW-521 Input Parameter Check

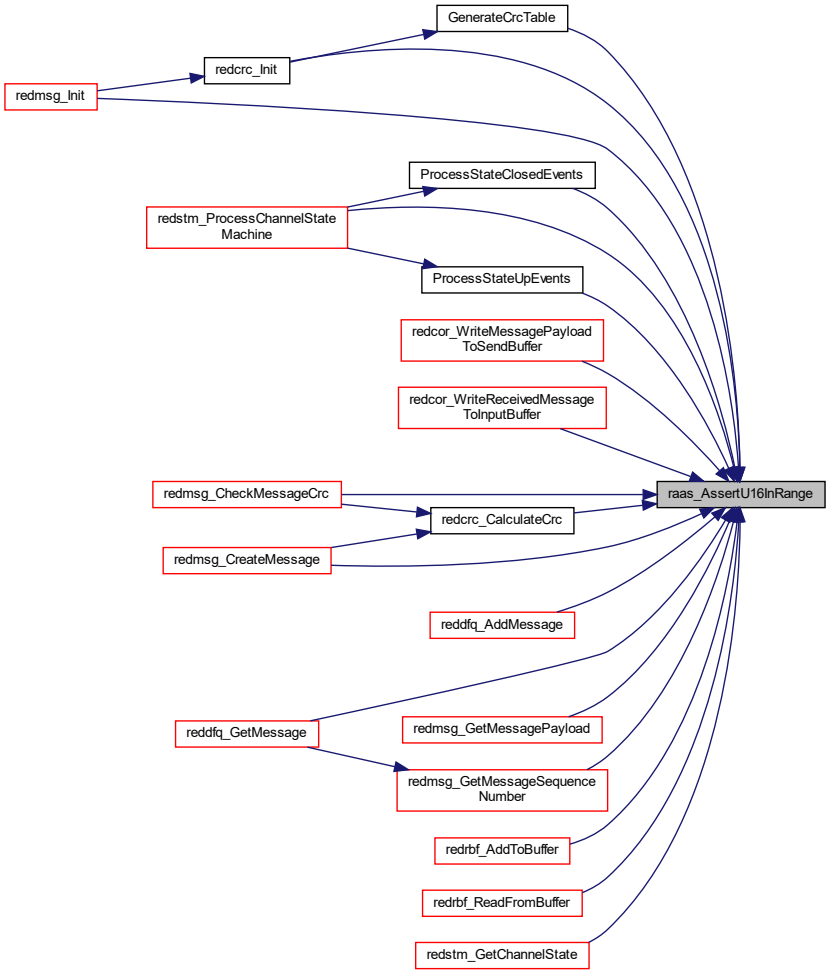
Parameters

in	*value*	Value to check. The full value range is valid and usable.
in	*min_value*	Minimum value for the check. The full value range is valid and usable.
in	*max_value*	Maximum value for the check. The full value range is valid and usable.
in	*error_reason*	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:



4.18.2.4 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.

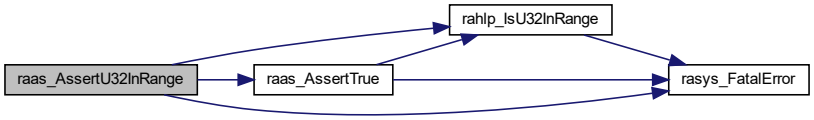
This function checks if a uint32_t value is inside a defined range. Check condition: min_value <= value <= max↩ _value. The minimum value must be smaller or equal to the maximum value, otherwise a radef_kInvalidParameter fatal error is thrown.

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

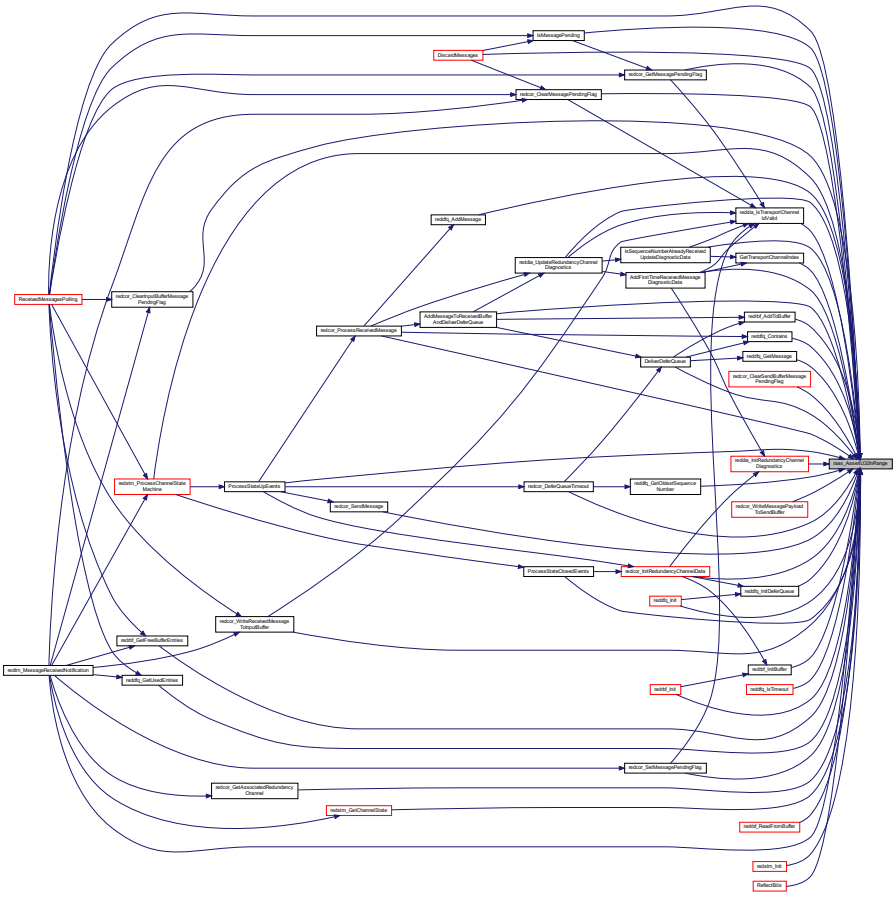
Parameters

in	*value*	Value to check. The full value range is valid and usable.
in	*min_value*	Minimum value for the check. The full value range is valid and usable.
in	*max_value*	Maximum value for the check. The full value range is valid and usable.
in	*error_reason*	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:



4.18.2.5 raas_AssertU8InRange()

```
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.

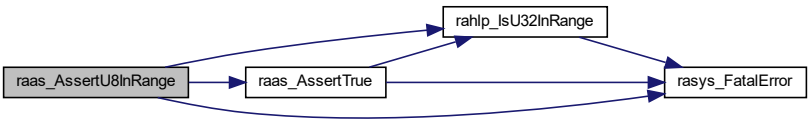
This function checks if a uint8_t value is inside a defined range. Check condition: min_value <= value <= max_value. The minimum value must be smaller or equal to the maximum value, otherwise a radef_kInvalidParameter fatal error is thrown.

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

Parameters

in	*value*	Value to check. The full value range is valid and usable.
in	*min_value*	Minimum value for the check. The full value range is valid and usable.
in	*max_value*	Maximum value for the check. The full value range is valid and usable.
in	*error_reason*	Reason in case of error. Valid range: radef_kMin <= value < radef_kMax.

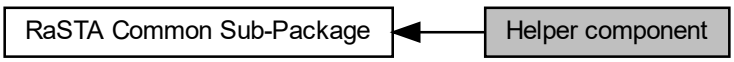
Here is the call graph for this function:



4.19 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.

4.19.1 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

4.19.2 Function Documentation

4.19.2.1 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

in	*value*	Value to check. The full value range is valid and usable.
in	*min_value*	Minimum value for the check. The full value range is valid and usable.
in	*max_value*	Maximum value for the check. The full value range is valid and usable.

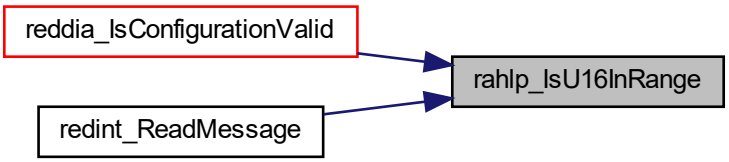
Returns

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:



4.19.2.2 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	*value*	Value to check. The full value range is valid and usable.
in	*min_value*	Minimum value for the check. The full value range is valid and usable.
in	*max_value*	Maximum value for the check. The full value range is valid and usable.

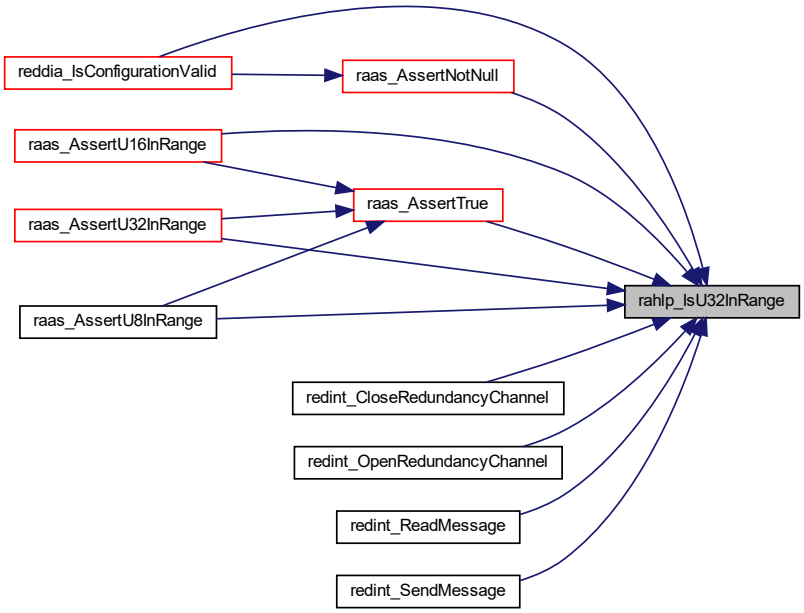
Returns

true -> successful operation, value inside the range

false -> value outside the range

Here is the call graph for this function:

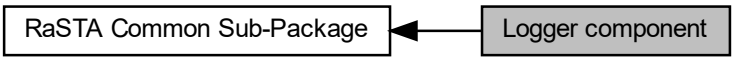
Here is the caller graph for this function:



4.20 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.

4.20.1 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 form 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

4.20.2 Macro Definition Documentation

4.20.2.1 ralog_ENABLE_LOGGER `#define ralog_ENABLE_LOGGER 0`

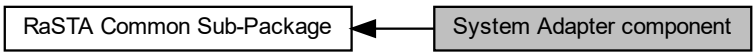
Global logger disable.

ralog_ENABLE_LOGGER 0 -> logger disabled, used for RELEASE build, no logger code is generated

4.21 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.

4.21.1 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

4.21.2 Function Documentation

4.21.2.1 rasys_FatalError() `void rasys_FatalError ( const radef_RaStaReturnCode error_reason )`

Fatal error function.

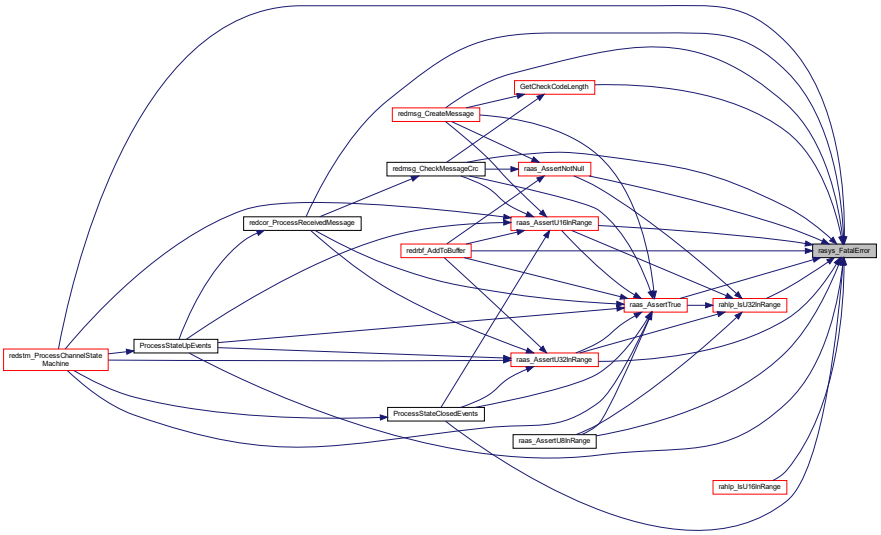
This function returns the program execution to the operating system. This function is called in case of a fatal internal error. Important: This function is not allowed to return.

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	*error_reason*	Reason of the fatal error. Valid range: radef_kMin <= value < radef_kMax.

Here is the caller graph for this function:



4.21.2.2 rasys_GetRandomNumber() `uint32_t rasys_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

4.21.2.3 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].

4.21.2.4 rasys_GetTimerValue() `uint32_t rasys_GetTimerValue ( void )`

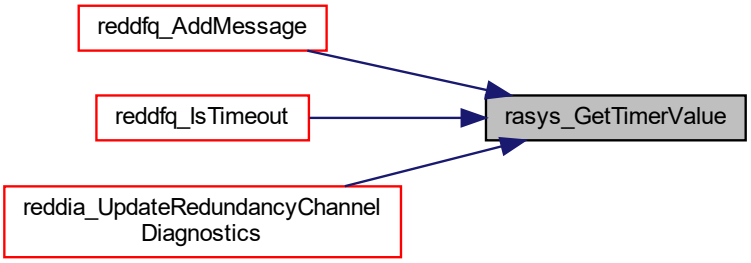
Returns the actual value of a free running up counting timer.

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 CrcOptions Struct Reference

Typedef for the options of the CRC algorithm.

Public Attributes

- uint16_t **width**

  Length of CRC [bit].
- uint32_t **polynomial**

  The CRC polynomial without MSB.
- uint32_t **initial_optimized**

  The initial value for the table lookup algorithm.
- bool **refin**

  true, if reflected input is enabled
- bool **refout**

  true, if reflected output is enabled
- uint32_t **final_xor**

  Value for the final xor operation, hast to be the same length as width.

5.1.1 Detailed Description

Typedef for the options of the CRC algorithm.

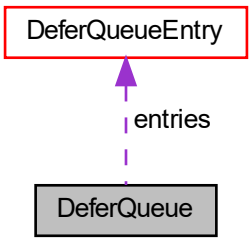
The documentation for this struct was generated from the following file:

- redcrc_red_crc.c

5.2 DeferQueue Struct Reference

Typedef for a defer queue.

Collaboration diagram for DeferQueue:



Public Attributes

- uint32_t **used_defer_queue_entries**

  Number of used defer queue entries [messages].
- DeferQueueEntry **entries** [RADEF_MAX_DEFER_QUEUE_SIZE]

  Array containing all defer queue entries.

5.2.1 Detailed Description

Typedef for a defer queue.

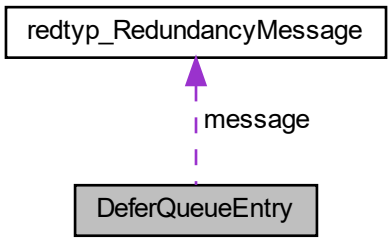
The documentation for this struct was generated from the following file:

- reddfq_red_defer_queue.c

5.3 DeferQueueEntry Struct Reference

Typedef for a defer queue entry.

Collaboration diagram for DeferQueueEntry:



Public Attributes

- redtyp_RedundancyMessage **message**
  Redundancy layer PDU message.
- uint32_t **received_timestamp**
  Message received timestamp [ms].

5.3.1 Detailed Description

Typedef for a defer queue entry.

The documentation for this struct was generated from the following file:

- reddfq_red_defer_queue.c

5.4 radef_TransportChannelDiagnosticData Struct Reference

Struct for the diagnostic data from a transport channel.

`#include <radef_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.

5.4.1 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

5.4.2 Member Data Documentation

5.4.2.1 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

5.4.2.2 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

5.4.2.3 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

5.4.2.4 t_drift2 `uint32_t radef_TransportChannelDiagnosticData::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

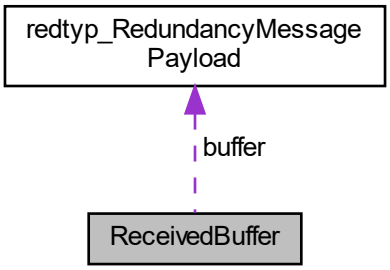
The documentation for this struct was generated from the following file:

- radef_rasta_definitions.h

5.5 ReceivedBuffer Struct Reference

Struct for redundancy layer received messages payload buffer.

Collaboration diagram for ReceivedBuffer:



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 [messages]
- redtyp_RedundancyMessagePayload buffer [RADEF_MAX_N_SEND_MAX]

5.5.1 Detailed Description

Struct for redundancy layer received messages payload buffer.

5.5.2 Member Data Documentation

5.5.2.1 buffer `redtyp_RedundancyMessagePayload ReceivedBuffer::buffer[RADEF_MAX_N_SEND_MAX]`

buffer with the payload of correctly received messages, waiting for the read from the safety and retransmission layer adapter

The documentation for this struct was generated from the following file:

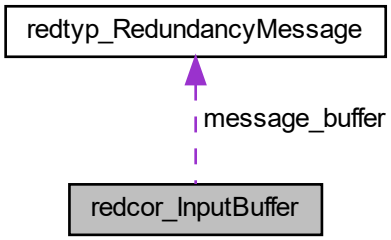
- redrbf_red_received_buffer.c

5.6 redcor_InputBuffer Struct Reference

Struct for the newly received message input buffer.

`#include <redcor_red_core.h>`

Collaboration diagram for redcor_InputBuffer:



Public Attributes

- bool **message_in_buffer**

  flag which indicates, that a new unprocessed message is in the message input buffer
- uint32_t **transport_channel_id**

  transport channel of message in input buffer, used for diagnostics
- redtyp_RedundancyMessage **message_buffer**

  input buffer for newly received message

5.6.1 Detailed Description

Struct for the newly received message input buffer.

The documentation for this struct was generated from the following file:

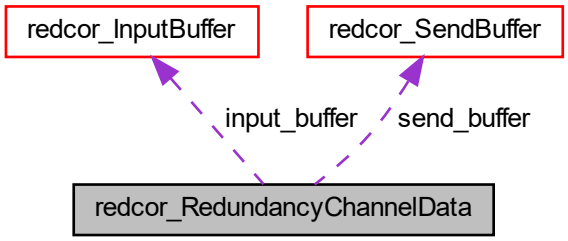
- redcor_red_core.h

5.7 redcor_RedundancyChannelData Struct Reference

Struct for the process data of a redundancy channel.

`#include <redcor_red_core.h>`

Collaboration diagram for redcor_RedundancyChannelData:



Public Attributes

- uint32_t **seq_tx**
  next sequence number to be sent
- uint32_t **seq_rx**
  next sequence number expected for the receipt
- bool **received_data_pending** [RADEF_MAX_NUMBER_OF_RED_CHANNELS ∗RADEF_MAX_NUMBER_OF_TRANSPORT
  true, if received data is pending on a transport channel
- redcor_InputBuffer **input_buffer**
  input buffer for newly received message
- redcor_SendBuffer **send_buffer**
  buffer for message payload to send

5.7.1 Detailed Description

Struct for the process data of a redundancy channel.

The documentation for this struct was generated from the following file:

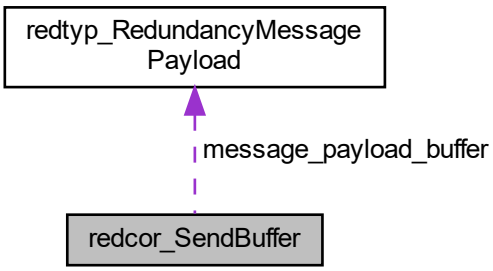
- redcor_red_core.h

5.8 redcor_SendBuffer Struct Reference

Struct for the message payload send buffer.

`#include <redcor_red_core.h>`

Collaboration diagram for redcor_SendBuffer:



Public Attributes

- bool **message_in_buffer**

  flag which indicates, that a unprocessed message is in the send buffer
- redtyp_RedundancyMessagePayload **message_payload_buffer**

  buffer for message payload to send

5.8.1 Detailed Description

Struct for the message payload send buffer.

The documentation for this struct was generated from the following file:

- redcor_red_core.h

5.9 redcty_RedundancyChannelConfiguration Struct Reference

Struct for the configuration data of a redundancy channel.

`#include <redcty_red_config_types.h>`

Public Attributes

- uint32_t **red_channel_id**

  Redundancy channel identification. Valid range: 0 <= value < configured number of redundancy channels.
- uint32_t num_transport_channels
- uint32_t transport_channel_ids [RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS]

5.9.1 Detailed Description

Struct for the configuration data of a redundancy channel.

5.9.2 Member Data Documentation

5.9.2.1 num_transport_channels `uint32_t redcty_RedundancyChannelConfiguration::num_transport_channels`

Number of used transport channels in this redundancy channel. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_TRANSPOR

5.9.2.2 transport_channel_ids `uint32_t redcty_RedundancyChannelConfiguration::transport_channel_ids[RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS]`

IDs of the associated transport channels. Valid range: 0 <= value < RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS * RADEF_MAX_NUMBER_OF_RED_CHANNELS

The documentation for this struct was generated from the following file:

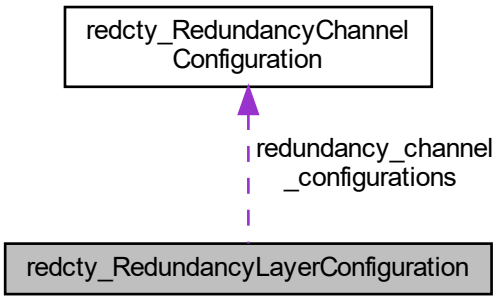
- redcty_red_config_types.h

5.10 redcty_RedundancyLayerConfiguration Struct Reference

Struct for the configuration data of the redundancy layer.

`#include <redcty_red_config_types.h>`

Collaboration diagram for redcty_RedundancyLayerConfiguration:



Public Attributes

- redcty_CheckCodeType **check_code_type**

  Type of check code [enum]. All enum entries of redcty_CheckCodeType are valid and usable.
- uint32_t **t_seq**

  Time for out of sequence message buffering (Tseq) [ms]. Valid range: redcty_kMinTSeq <= value <= redcty_kMaxTSeq.
- uint32_t **n_diagnosis**

  RedL diagnosis window size [messages]. Valid range: redcty_kMinRedLayerNDiagnosis <= value <= RADEF_MAX_RED_LAYER_N_DIAGNOSIS.
- uint32_t **n_defer_queue_size**

  Size of defer queue [messages]. Valid range: redcty_kMinDeferQueueSize <= value <= RADEF_MAX_DEFER_QUEUE_SIZE.
- uint32_t number_of_redundancy_channels
- redcty_RedundancyChannelConfiguration **redundancy_channel_configurations** [RADEF_MAX_NUMBER_OF_RED_CHAN

  Redundancy channel configurations.

5.10.1 Detailed Description

Struct for the configuration data of the redundancy layer.

5.10.2 Member Data Documentation

5.10.2.1 number_of_redundancy_channels `uint32_t redcty_RedundancyLayerConfiguration::number↩_of_redundancy_channels`

Number of configured redundancy channels. Valid range: redcty_kMinNumberOfRedundancyChannels <= value <= RADEF_MAX_NUMBER_OF_RED_CHANNELS

The documentation for this struct was generated from the following file:

- redcty_red_config_types.h

5.11 reddia_ReceivedMessageTimestamp Struct Reference

Struct for the timestamps of first received messages.

`#include <reddia_red_diagnostics.h>`

Public Attributes

- uint32_t **sequence_number**

  Sequence number of the message.
- uint32_t **received_time_stamp**

  Message received timestamp, when this message was received first.
- bool message_received_flag [RADEF_MAX_NUMBER_OF_TRANSPORT_CHANNELS]

5.11.1 Detailed Description

Struct for the timestamps of first received messages.

5.11.2 Member Data Documentation

5.11.2.1 message_received_flag `bool reddia_ReceivedMessageTimestamp::message_received_flag[RADEF_MAX_NUMBER_OF`

Message received flag for each transport channel. This is used to correctly count n_missed, for messages which didn't arrive at all on a transport channel.

The documentation for this struct was generated from the following file:

- reddia_red_diagnostics.h

5.12 redtyp_RedundancyMessage Struct Reference

Typedef for a redundancy layer PDU message.

`#include <redtyp_red_types.h>`

Public Attributes

- uint16_t **message_size**

  Used message size [bytes]. Valid range: RADEF_MIN_RED_LAYER_PDU_MESSAGE_SIZE <= value <= RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE.
- uint8_t **message** [RADEF_MAX_RED_LAYER_PDU_MESSAGE_SIZE]

  Message buffer. For the message data the full value range is valid and usable.

5.12.1 Detailed Description

Typedef for a redundancy layer PDU message.

The documentation for this struct was generated from the following file:

- redtyp_red_types.h

5.13 redtyp_RedundancyMessagePayload Struct Reference

Typedef for a redundancy layer PDU message payload.

`#include <redtyp_red_types.h>`

Public Attributes

- uint16_t **payload_size**

  Used payload size [bytes]. Valid range: RADEF_SR_LAYER_MESSAGE_HEADER_SIZE <= value <= RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE.
- uint8_t **payload** [RADEF_MAX_SR_LAYER_PDU_MESSAGE_SIZE]

  Payload buffer. For the payload data the full value range is valid and usable.

5.13.1 Detailed Description

Typedef for a redundancy layer PDU message payload.

The documentation for this struct was generated from the following file:

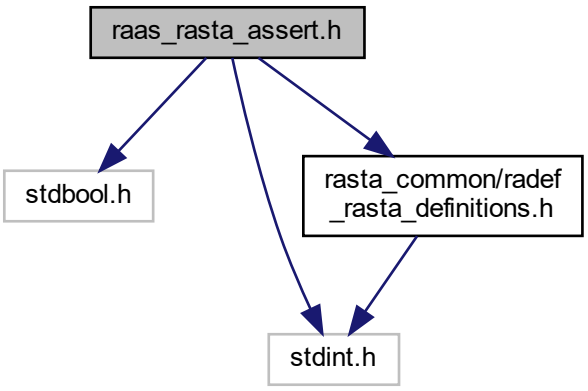
- redtyp_red_types.h

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

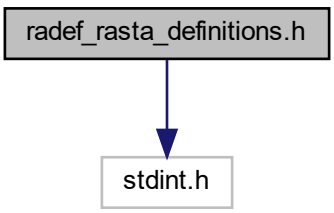
64779fea6efa1199e3a82cbff64181dea3877e8d

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_IN - 1U)

  Size of timing distribution diagnostic interval array. Contains one element less then intervals since last element is set to t_max.
- #define **RADEF_MAX_N_SEND_MAX** (20U)

  Maximum number of entires 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

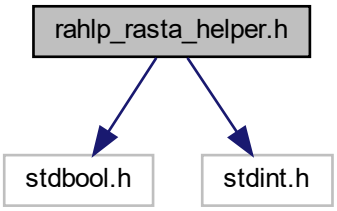
64779fea6efa1199e3a82cbff64181dea3877e8d

Changelog **-:** Initial version (-, -)

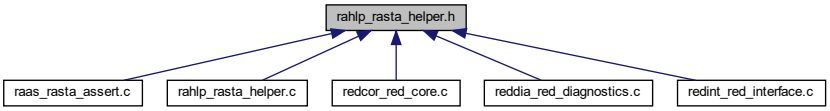
6.3 rahlp_rasta_helper.h File Reference

```
#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

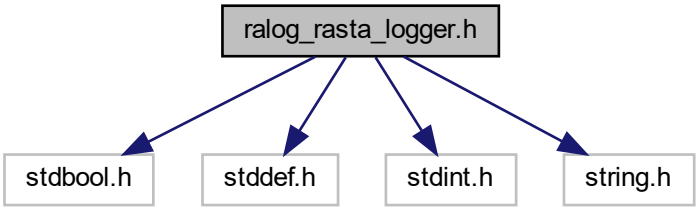
64779fea6efa1199e3a82cbff64181dea3877e8d

Changelog **-:** Initial version (-, -)

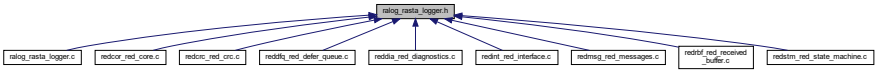
6.4 ralog_rasta_logger.h File Reference

```
#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

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

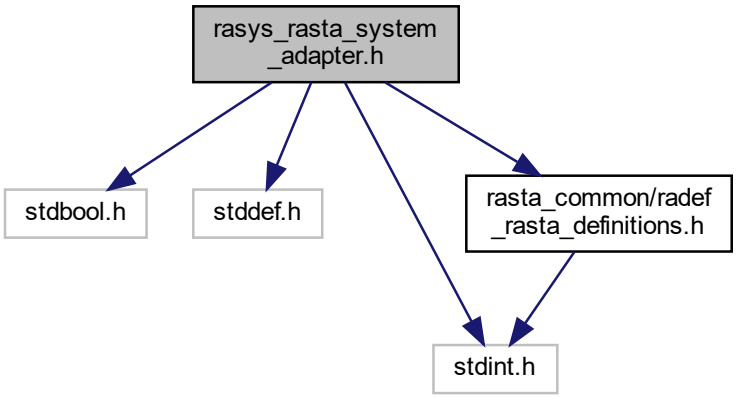
64779fea6efa1199e3a82cbff64181dea3877e8d

Changelog **-:** Initial version (-, -)

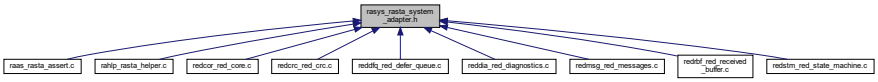
6.5 rasys_rasta_system_adapter.h File Reference

```
#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

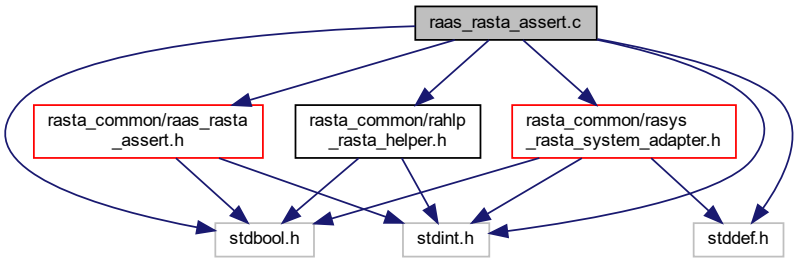
Changelog **-:** Initial version (-, -)

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

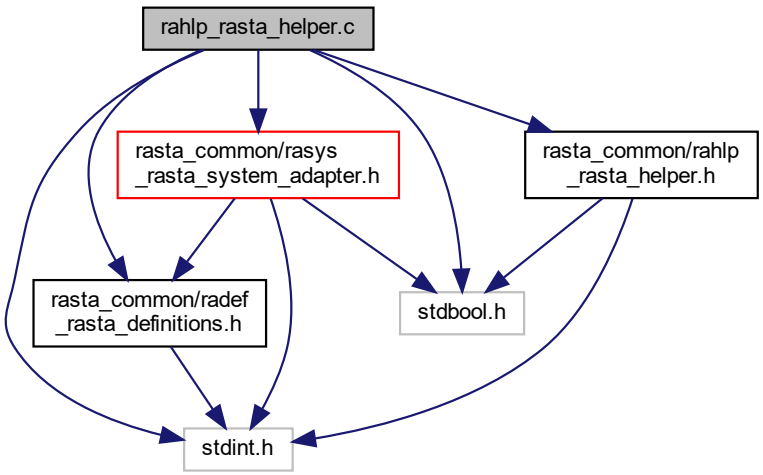
Changelog **-:** Initial version (-, -)

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

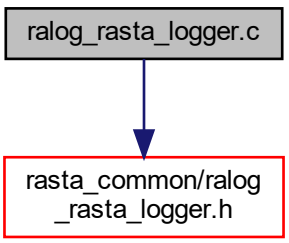
Changelog **-:** Initial version (-, -)

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

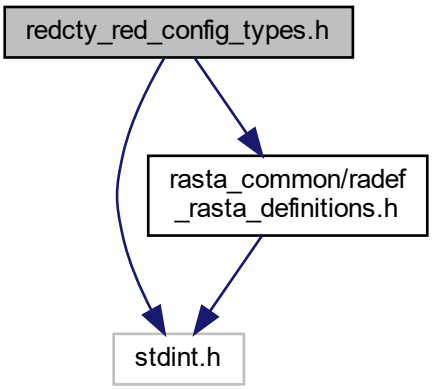
64779fea6efa1199e3a82cbff64181dea3877e8d

Changelog **-:** Initial version (-, -)

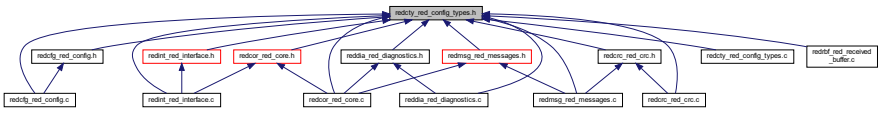
6.9 redcty_red_config_types.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
```

Include dependency graph for redcty_red_config_types.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct redcty_RedundancyChannelConfiguration

  Struct for the configuration data of a redundancy channel.
- struct redcty_RedundancyLayerConfiguration

  Struct for the configuration data of the redundancy layer.

Enumerations

- enum redcty_CheckCodeType {
  redcty_kCheckCodeMin = 0 , redcty_kCheckCodeA = 0 , redcty_kCheckCodeB = 1 , redcty_kCheckCodeC = 2 ,
  redcty_kCheckCodeD = 3 , redcty_kCheckCodeE = 4 , redcty_kCheckCodeMax }

  Enum for the check code type of the redundancy channels.

Variables

- const uint32_t **redcty_kMinNumberOfRedundancyChannels**

  Minimum number of redundancy channels.
- const uint32_t **redcty_kMinNumberOfTransportChannels**

  Minimum number of transport channels.
- const uint32_t **redcty_kMinTSeq**

  Minimum time for out of sequence message buffering (Tseq) [ms].
- const uint32_t **redcty_kMaxTSeq**

  Maximum time for out of sequence message buffering (Tseq) [ms].
- const uint32_t **redcty_kMinRedLayerNDiagnosis**

  Minimum diagnosis window size [messages].
- const uint32_t **redcty_kMinDeferQueueSize**

  Minimum size of a redundancy channel defer queue [messages].

6.9.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

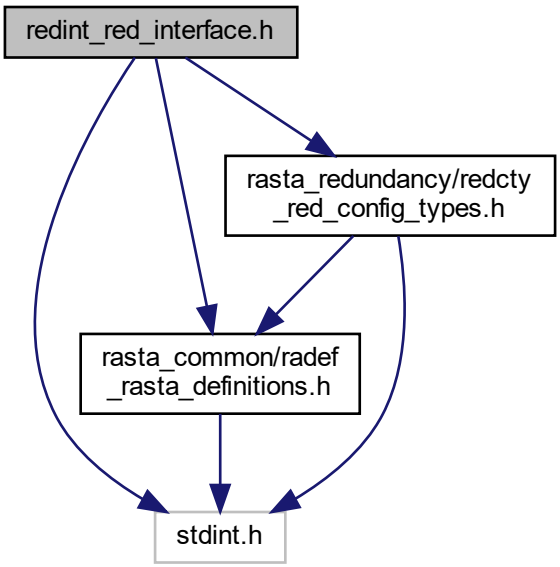
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

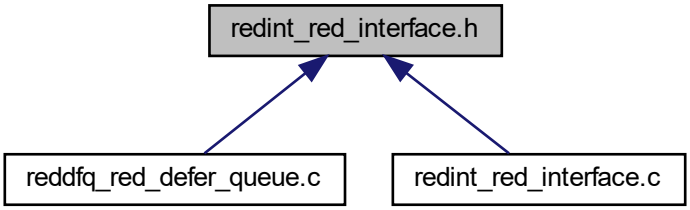
6.10 redint_red_interface.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_redundancy/redcty_red_config_types.h"
```

Include dependency graph for redint_red_interface.h:



This graph shows which files directly or indirectly include this file:



Functions

- radef_RaStaReturnCode redint_Init (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Initialize the RedL.
- radef_RaStaReturnCode redint_GetInitializationState (void)

Get the initialization state of the RedL.

- radef_RaStaReturnCode redint_OpenRedundancyChannel (const uint32_t redundancy_channel_id)

  Open a given redundancy channel.
- radef_RaStaReturnCode redint_CloseRedundancyChannel (const uint32_t redundancy_channel_id)

  Close a given redundancy channel.
- radef_RaStaReturnCode redint_SendMessage (const uint32_t redundancy_channel_id, const uint16_t message_size, const uint8_t *const message_data)

  Send a message over a given redundancy channel.
- radef_RaStaReturnCode redint_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 given redundancy channel.
- radef_RaStaReturnCode redint_CheckTimings (void)

  Check redundancy layer timings and read pending messages form the transport channels.

6.10.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

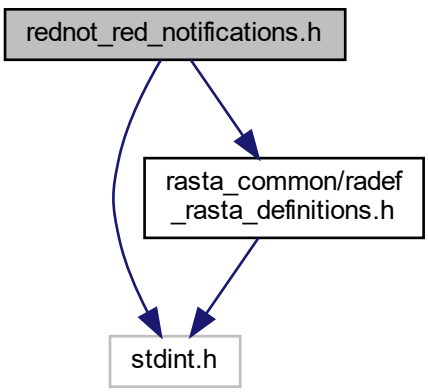
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

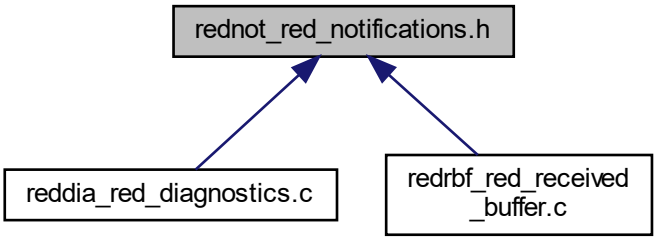
6.11 rednot_red_notifications.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
```

Include dependency graph for rednot_red_notifications.h:



This graph shows which files directly or indirectly include this file:



Functions

- void rednot_MessageReceivedNotification (const uint32_t red_channel_id)

  Redundancy layer message received notification function to SafRetL adapter.
- void rednot_DiagnosticNotification (const uint32_t red_channel_id, const uint32_t tr_channel_id, const radef_TransportChannelDiagnosticData TransportChannelDiagnosticData)

  Redundancy layer diagnostic notification function to SafRetL adapter.

6.11.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

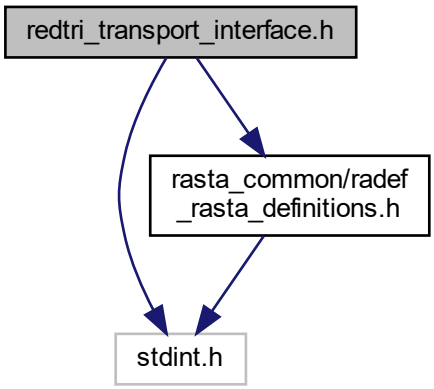
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

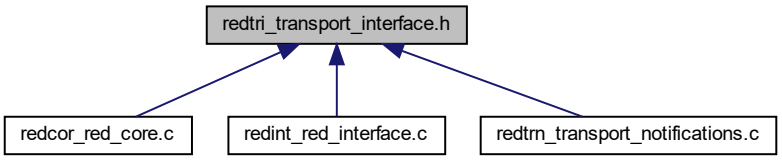
6.12 redtri_transport_interface.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
```

Include dependency graph for redtri_transport_interface.h:



This graph shows which files directly or indirectly include this file:



Functions

- void redtri_Init (void)

  Initialize transport layer.
- void redtri_SendMessage (const uint32_t transport_channel_id, const uint16_t message_size, const uint8_t ∗const message_data)

  Send a RedL message over a transport channel.
- radef_RaStaReturnCode redtri_ReadMessage (const uint32_t transport_channel_id, const uint16_t buffer↩ _size, uint16_t ∗const message_size, uint8_t ∗const message_buffer)

  Read a received RedL message from a transport channel.

6.12.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

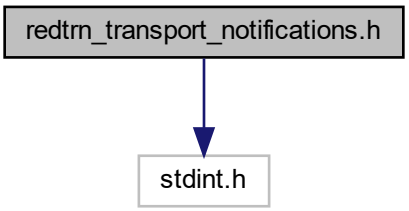
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

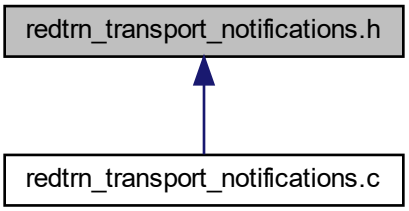
6.13 redtrn_transport_notifications.h File Reference

```
#include <stdint.h>
```

Include dependency graph for redtrn_transport_notifications.h:



This graph shows which files directly or indirectly include this file:



Functions

- void redtrn_MessageReceivedNotification (const uint32_t transport_channel_id)

  Transport layer message received notification function.

6.13.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

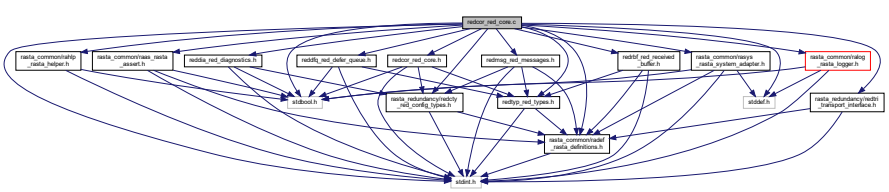
Changelog **-:** Initial version (-, -)

6.14 redcor_red_core.c File Reference

Implementation of RaSTA redundancy layer core module.

```
#include "redcor_red_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_redundancy/redcty_red_config_types.h"
#include "rasta_redundancy/redtri_transport_interface.h"
#include "reddfq_red_defer_queue.h"
#include "reddia_red_diagnostics.h"
#include "redmsg_red_messages.h"
#include "redrbf_red_received_buffer.h"
#include "redtyp_red_types.h"
```

Include dependency graph for redcor_red_core.c:



Functions

- static void DeliverDeferQueue (const uint32_t red_channel_id)

  This function delivers the messages from the defer queue to the received buffer.
- static void AddMessageToReceivedBufferAndDeliverDeferQueue (const uint32_t red_channel_id)

  Add a received message to the received buffer and call the DeliverDeferQueue() function.
- bool redcor_IsConfigurationValid (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Checks if the redundancy layer configuration is valid.
- void redcor_Init (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Initialize all data of the redundancy layer core.
- void redcor_InitRedundancyChannelData (const uint32_t red_channel_id)

  Initialize the data of a dedicated redundancy channel.
- void redcor_DeferQueueTimeout (const uint32_t red_channel_id)

  Handle the defer queue timeout and deliver the defer queue messages to the received buffer.
- void redcor_WriteReceivedMessageToInputBuffer (const uint32_t red_channel_id, const uint32_t transport↩ _channel_id, const redtyp_RedundancyMessage *const received_message)

  Write a received message to the input buffer.
- void redcor_ClearInputBufferMessagePendingFlag (const uint32_t red_channel_id)

  Clear input buffer message pending flag.
- void redcor_ProcessReceivedMessage (const uint32_t red_channel_id)

  Process a received message from the input buffer.
- void redcor_SetMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Set a flag, which indicates that a received message is pending to read from the transport layer.
- bool redcor_GetMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Get the received message pending flag for a dedicated transport channel.
- void redcor_ClearMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Clear the received message pending flag.
- void redcor_WriteMessagePayloadToSendBuffer (const uint32_t red_channel_id, const uint16_t payload_↩ size, const uint8_t *const payload_data)

  Write message payload to send buffer.
- void redcor_ClearSendBufferMessagePendingFlag (const uint32_t red_channel_id)

  Clear send buffer message pending flag.
- void redcor_SendMessage (const uint32_t red_channel_id)

  Send a redundancy layer message from the send buffer to the transport channels.
- void redcor_GetAssociatedRedundancyChannel (const uint32_t transport_channel_id, uint32_t *const red↩ _channel_id)

  Get the associated redundancy channel from a given transport channel.

Variables

- PRIVATE bool **redcor_initialized** = false

  Initialization state of the module. True, if the module is initialized.
- PRIVATE const redcty_RedundancyLayerConfiguration * **redcor_redundancy_configuration** = NULL

  Pointer to redundancy layer configuration.
- PRIVATE redcor_RedundancyChannelData **redcor_redundancy_channels** [RADEF_MAX_NUMBER_OF_RED_CHANNELS

  Process data of the redundancy channels.
- PRIVATE uint16_t **redcor_logger_id**

  ID of the redundancy core debug logger.
- static const uint32_t **kSequenceNumberRangeCheckFactor** = 10U

  Factor for sequence number validity range check.

6.14.1 Detailed Description

Implementation of RaSTA redundancy layer core module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

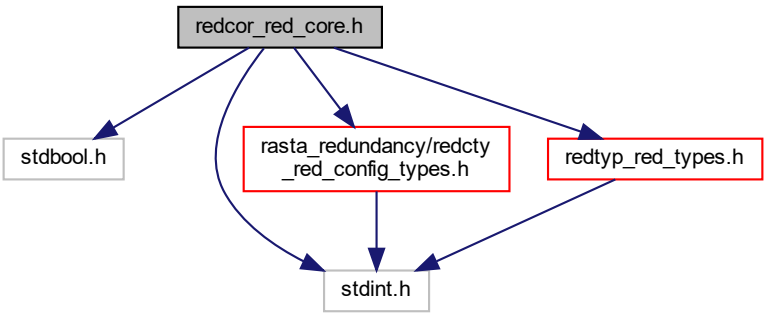
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

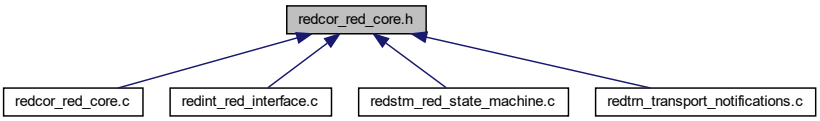
6.15 redcor_red_core.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "rasta_redundancy/redcty_red_config_types.h"
#include "redtyp_red_types.h"
```

Include dependency graph for redcor_red_core.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct redcor_InputBuffer

  Struct for the newly received message input buffer.
- struct redcor_SendBuffer

  Struct for the message payload send buffer.
- struct redcor_RedundancyChannelData

  Struct for the process data of a redundancy channel.

Functions

- bool redcor_IsConfigurationValid (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Checks if the redundancy layer configuration is valid.
- void redcor_Init (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Initialize all data of the redundancy layer core.
- void redcor_InitRedundancyChannelData (const uint32_t red_channel_id)

  Initialize the data of a dedicated redundancy channel.
- void redcor_DeferQueueTimeout (const uint32_t red_channel_id)

  Handle the defer queue timeout and deliver the defer queue messages to the received buffer.
- void redcor_WriteReceivedMessageToInputBuffer (const uint32_t red_channel_id, const uint32_t transport↩ _channel_id, const redtyp_RedundancyMessage *const received_message)

  Write a received message to the input buffer.
- void redcor_ClearInputBufferMessagePendingFlag (const uint32_t red_channel_id)

  Clear input buffer message pending flag.
- void redcor_ProcessReceivedMessage (const uint32_t red_channel_id)

  Process a received message from the input buffer.
- void redcor_SetMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Set a flag, which indicates that a received message is pending to read from the transport layer.
- bool redcor_GetMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Get the received message pending flag for a dedicated transport channel.
- void redcor_ClearMessagePendingFlag (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Clear the received message pending flag.
- void redcor_WriteMessagePayloadToSendBuffer (const uint32_t red_channel_id, const uint16_t payload_↩ size, const uint8_t *const payload_data)

  Write message payload to send buffer.
- void redcor_ClearSendBufferMessagePendingFlag (const uint32_t red_channel_id)

  Clear send buffer message pending flag.
- void redcor_SendMessage (const uint32_t red_channel_id)

  Send a redundancy layer message from the send buffer to the transport channels.
- void redcor_GetAssociatedRedundancyChannel (const uint32_t transport_channel_id, uint32_t *const red↩ _channel_id)

  Get the associated redundancy channel from a given transport channel.

6.15.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

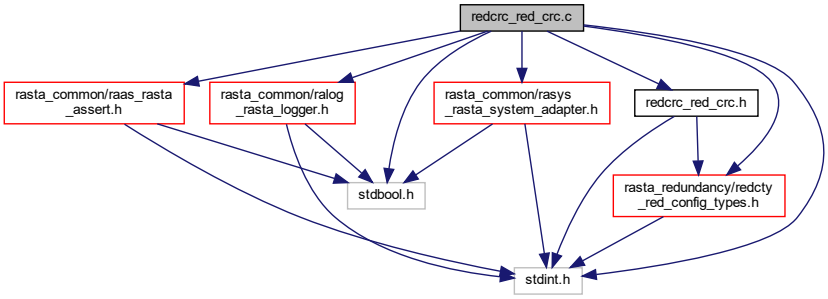
Changelog **-:** Initial version (-, -)

6.16 redcrc_red_crc.c File Reference

Implementation of RaSTA redundancy layer CRC module.

```
#include "redcrc_red_crc.h"
#include <stdbool.h>
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/ralog_rasta_logger.h"
#include "rasta_common/rasys_rasta_system_adapter.h"
#include "rasta_redundancy/redcty_red_config_types.h"
```

Include dependency graph for redcrc_red_crc.c:



Classes

- struct CrcOptions

  Typedef for the options of the CRC algorithm.

Macros

- #define **CRC_TABLE_SIZE** (256U)

  Size of the CRC lookup table [elements].

Functions

- static uint32_t ReflectBits (uint32_t value_in, uint16_t number_of_bits)

  Reflects the lower number_of_bits of a uint32_t and returns a value containing the reflected bits.
- static void GenerateCrcTable (void)

  Generates a CRC lookup table according to the configured redcrc_check_code_type.
- void redcrc_Init (const redcty_CheckCodeType configured_check_code_type)

  Initialize the CRC module and generate the CRC lookup table according to the configured check_code_type.
- void redcrc_CalculateCrc (const uint16_t data_size, const uint8_t *const data_buffer, uint32_t *const calculated_crc)

  Calculate the defined type of CRC of a data buffer.

Variables

- PRIVATE bool **redcrc_initialized** = false

  Initialization state of the module. True, if the module is initialized.
- PRIVATE redcty_CheckCodeType **redcrc_check_code_type**

  Configured check code type.
- PRIVATE uint32_t **redcrc_table** [CRC_TABLE_SIZE]

  The CRC lookup table, calculated by calling GenerateCrcTable().
- PRIVATE uint32_t **redcrc_crc_mask**

  CRC mask for internal CRC computation.
- PRIVATE uint32_t **redcrc_crc_high_bit**

  CRC high bit for internal CRC computation.
- static const uint16_t **kMinWidth** = 8U

  Minimum supported CRC width [bits].
- static const uint16_t **kMaxWidth** = 32U

  Maximum supported CRC width [bits].
- static const uint16_t **kBitsPerByte** = 8U

  Bits per byte [bits].
- static const CrcOptions **kCrcOptions** [(uint16_t) redcty_kCheckCodeMax]

  CRC options for all check code types.

6.16.1 Detailed Description

Implementation of RaSTA redundancy layer CRC module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

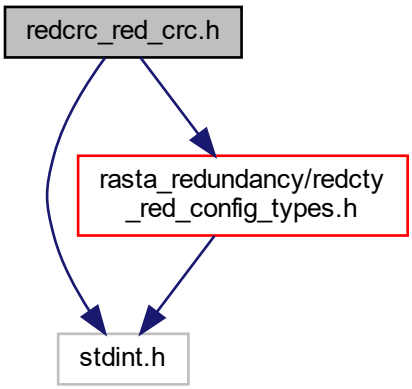
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

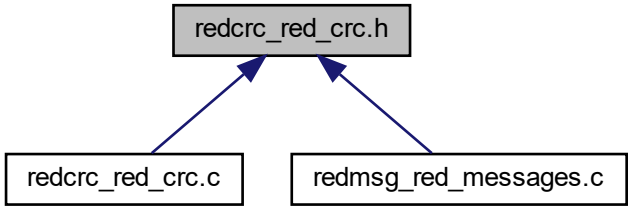
6.17 redcrc_red_crc.h File Reference

```
#include <stdint.h>
#include "rasta_redundancy/redcty_red_config_types.h"
```

Include dependency graph for redcrc_red_crc.h:



This graph shows which files directly or indirectly include this file:



Functions

- void redcrc_Init (const redcty_CheckCodeType configured_check_code_type)

  Initialize the CRC module and generate the CRC lookup table according to the configured check_code_type.
- void redcrc_CalculateCrc (const uint16_t data_size, const uint8_t ∗const data_buffer, uint32_t ∗const calculated_crc)

  Calculate the defined type of CRC of a data buffer.

6.17.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

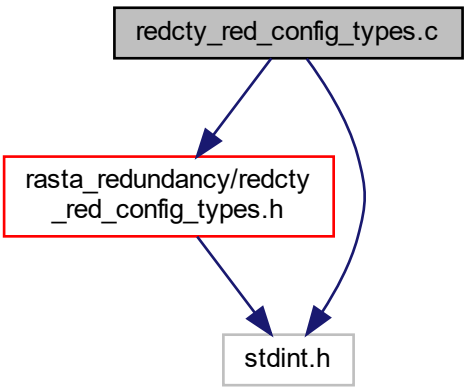
Changelog **-:** Initial version (-, -)

6.18 redcty_red_config_types.c File Reference

Definition of RaSTA redundancy layer configuration min./max. range constants.

```
#include "rasta_redundancy/redcty_red_config_types.h"
#include <stdint.h>
```

Include dependency graph for redcty_red_config_types.c:



Variables

- const uint32_t **redcty_kMinNumberOfRedundancyChannels** = 1U
  Minimum number of redundancy channels.
- const uint32_t **redcty_kMinNumberOfTransportChannels** = 1U
  Minimum number of transport channels.
- const uint32_t **redcty_kMinTSeq** = 50U
  Minimum time for out of sequence message buffering (Tseq) [ms].
- const uint32_t **redcty_kMaxTSeq** = 500U
  Maximum time for out of sequence message buffering (Tseq) [ms].
- const uint32_t **redcty_kMinRedLayerNDiagnosis** = 10U
  Minimum diagnosis window size [messages].
- const uint32_t **redcty_kMinDeferQueueSize** = 4U
  Minimum size of a redundancy channel defer queue [messages].

6.18.1 Detailed Description

Definition of RaSTA redundancy layer configuration min./max. range constants.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

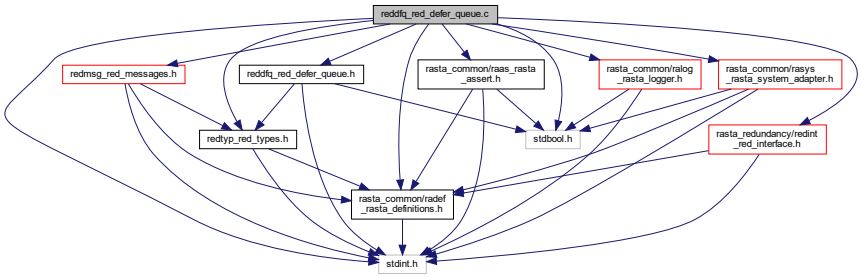
Changelog **-:** Initial version (-, -)

6.19 reddfq_red_defer_queue.c File Reference

Implementation of RaSTA redundancy layer defer queue module.

```
#include "reddfq_red_defer_queue.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_redundancy/redint_red_interface.h"
#include "redmsg_red_messages.h"
#include "redtyp_red_types.h"
```

Include dependency graph for reddfq_red_defer_queue.c:



Classes

- struct DeferQueueEntry

  Typedef for a defer queue entry.
- struct DeferQueue

  Typedef for a defer queue.

Functions

- void reddfq_Init (const uint32_t configured_red_channels, const uint32_t configured_defer_queue_size, const uint32_t configured_t_seq)

  Initialization of the data of the RedL defer queue module.
- void reddfq_InitDeferQueue (const uint32_t red_channel_id)

  Initialization of the defer queue of a dedicated redundancy channel.
- void reddfq_AddMessage (const uint32_t red_channel_id, const redtyp_RedundancyMessage *const redundancy_message)

  Add a redundancy layer message to the defer queue. If the queue is full, the message will be ignored.
- void reddfq_GetMessage (const uint32_t red_channel_id, const uint32_t sequence_number, redtyp_RedundancyMessage *const redundancy_message)

  Get and remove a redundancy layer message from the defer queue.
- bool reddfq_IsTimeout (const uint32_t red_channel_id)

  Check defer queue timeout on a dedicated redundancy channel.
- bool reddfq_Contains (const uint32_t red_channel_id, const uint32_t sequence_number)

  Checks if a message with a defined sequence number is in the defer queue.
- uint32_t reddfq_GetOldestSequenceNumber (const uint32_t red_channel_id)

  Returns the oldest sequence number found in the defer queue.
- uint32_t reddfq_GetUsedEntries (const uint32_t red_channel_id)

  Get the number of used defer queue entries.
- bool reddfq_IsSequenceNumberOlder (const uint32_t sequence_number_to_compare, const uint32_t sequence_number_reference)

  Returns true, if the sequence_number_to_compare is older than sequence_number_reference.

Variables

- PRIVATE bool **reddfq_initialized** = false

  Initialization state of the module. True, if the module is initialized.
- PRIVATE uint32_t **reddfq_number_of_red_channels** = 0U

  Number of configured redundancy channels.
- PRIVATE uint32_t **reddfq_defer_queue_size** = 0U

  Configured defer queue size [messages].
- PRIVATE uint32_t **reddfq_t_seq** = 0U

  Configured defer time [ms].
- PRIVATE DeferQueue **reddfq_defer_queues** [RADEF_MAX_NUMBER_OF_RED_CHANNELS]

  Defer queues for all redundancy channels.

6.19.1 Detailed Description

Implementation of RaSTA redundancy layer defer queue module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

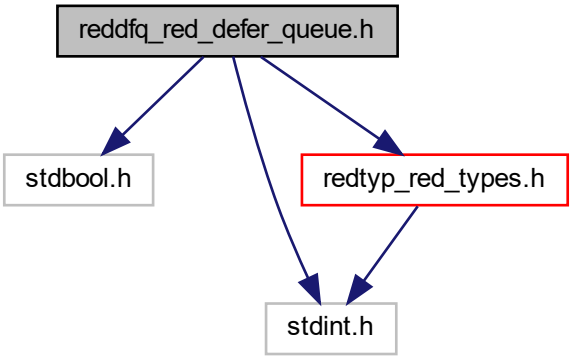
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

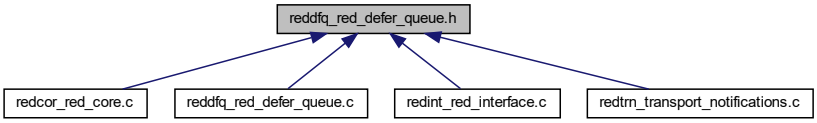
6.20 reddfq_red_defer_queue.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "redtyp_red_types.h"
```

Include dependency graph for reddfq_red_defer_queue.h:



This graph shows which files directly or indirectly include this file:



Functions

- void reddfq_Init (const uint32_t configured_red_channels, const uint32_t configured_defer_queue_size, const uint32_t configured_t_seq)

  Initialization of the data of the RedL defer queue module.
- void reddfq_InitDeferQueue (const uint32_t red_channel_id)

  Initialization of the defer queue of a dedicated redundancy channel.
- void reddfq_AddMessage (const uint32_t red_channel_id, const redtyp_RedundancyMessage *const redundancy_message)

  Add a redundancy layer message to the defer queue. If the queue is full, the message will be ignored.
- void reddfq_GetMessage (const uint32_t red_channel_id, const uint32_t sequence_number, redtyp_RedundancyMessage *const redundancy_message)

  Get and remove a redundancy layer message from the defer queue.
- bool reddfq_IsTimeout (const uint32_t red_channel_id)

  Check defer queue timeout on a dedicated redundancy channel.
- bool reddfq_Contains (const uint32_t red_channel_id, const uint32_t sequence_number)

Checks if a message with a defined sequence number is in the defer queue.

- uint32_t reddfq_GetOldestSequenceNumber (const uint32_t red_channel_id)

  Returns the oldest sequence number found in the defer queue.

- uint32_t reddfq_GetUsedEntries (const uint32_t red_channel_id)

  Get the number of used defer queue entries.

- bool reddfq_IsSequenceNumberOlder (const uint32_t sequence_number_to_compare, const uint32_t sequence_number_reference)

  Returns true, if the sequence_number_to_compare is older than sequence_number_reference.

6.20.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

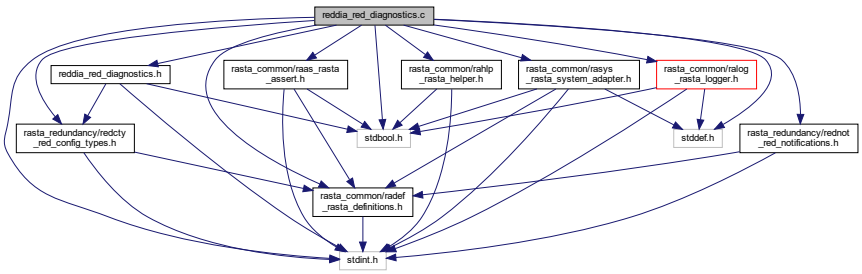
Changelog **-:** Initial version (-, -)

6.21 reddia_red_diagnostics.c File Reference

Implementation of RaSTA redundancy layer diagnostics.

```
#include "reddia_red_diagnostics.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_redundancy/redcty_red_config_types.h"
#include "rasta_redundancy/rednot_red_notifications.h"
```

Include dependency graph for reddia_red_diagnostics.c:



Functions

- uint32_t GetTransportChannelIndex (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Get the transport channel index of a specific transport channel of a redundancy channel.
- static bool IsSequenceNumberAlreadyReceivedUpdateDiagnosticData (const uint32_t red_channel_id, const uint32_t transport_channel_id, const uint32_t message_sequence_number, const uint32_t current_time_stamp)

  Returns true, if a received message timestamp of a message with the given sequence number is already stored. If so, it updates the diagnostic data of the transport channel of the newly received message.
- static void AddFirstTimeReceivedMessageDiagnosticData (const uint32_t red_channel_id, const uint32_t transport_channel_id, const uint32_t message_sequence_number, const uint32_t current_time_stamp)

  Add the diagnostic data of a message with a first time received sequence number and trigger the diagnostic notifications, if the diagnostic window is reached.
- void reddia_InitRedundancyLayerDiagnostics (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Initialize the RedL diagnostics module.
- void reddia_InitRedundancyChannelDiagnostics (const uint32_t red_channel_id)

  Initialize diagnostic data of a dedicated redundancy channel.
- void reddia_UpdateRedundancyChannelDiagnostics (const uint32_t red_channel_id, const uint32_t transport_channel_id, const uint32_t message_sequence_number)

  Update redundancy channel diagnostic data with the data of a newly received message.
- bool reddia_IsConfigurationValid (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Checks if the redundancy layer configuration is valid.
- bool reddia_IsTransportChannelIdValid (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Checks, if a transport channel identification is valid for a given redundancy channel.

Variables

- PRIVATE bool **reddia_initialized** = false

  Initialization state of the module. True, if the module is initialized.
- PRIVATE const redcty_RedundancyLayerConfiguration * **reddia_redundancy_configuration** = NULL

  Pointer to redundancy layer configuration.
- PRIVATE reddia_ReceivedMessageTimestamp **reddia_received_messages_timestamps** [RADEF_MAX_NUMBER_OF_RE

  Timestamps of earliest received messages.
- PRIVATE uint32_t **reddia_current_n_diagnosis** [RADEF_MAX_NUMBER_OF_RED_CHANNELS]

  Current number of messages in the current diagnosis window.
- PRIVATE radef_TransportChannelDiagnosticData **reddia_transport_channel_diagnostic_data** [RADEF_MAX_NUMBER_OF

  Diagnostic data of all transport channels.
- PRIVATE uint16_t **reddia_logger_id**

  ID of the redundancy diagnostics debug logger.

6.21.1 Detailed Description

Implementation of RaSTA redundancy layer diagnostics.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

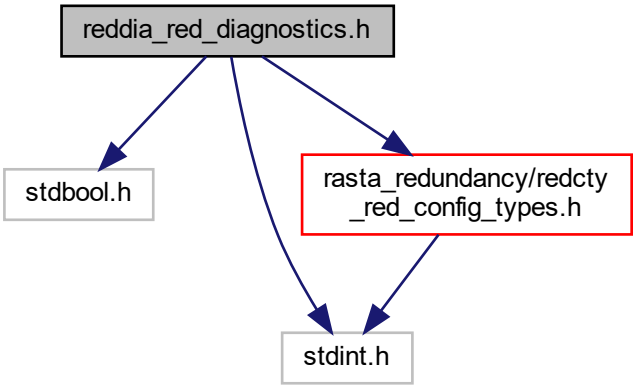
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

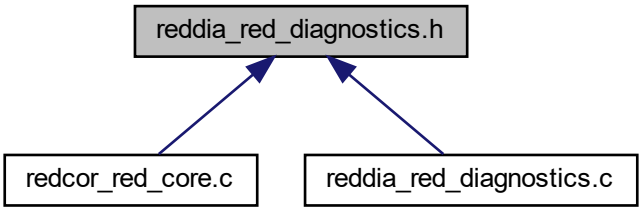
6.22 reddia_red_diagnostics.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "rasta_redundancy/redcty_red_config_types.h"
```

Include dependency graph for reddia_red_diagnostics.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct reddia_ReceivedMessageTimestamp

  Struct for the timestamps of first received messages.

Functions

- void reddia_InitRedundancyLayerDiagnostics (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

Initialize the RedL diagnostics module.

- void reddia_InitRedundancyChannelDiagnostics (const uint32_t red_channel_id)

  Initialize diagnostic data of a dedicated redundancy channel.
- void reddia_UpdateRedundancyChannelDiagnostics (const uint32_t red_channel_id, const uint32_t transport_channel_id, const uint32_t message_sequence_number)

  Update redundancy channel diagnostic data with the data of a newly received message.
- bool reddia_IsConfigurationValid (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Checks if the redundancy layer configuration is valid.
- bool reddia_IsTransportChannelIdValid (const uint32_t red_channel_id, const uint32_t transport_channel_id)

  Checks, if a transport channel identification is valid for a given redundancy channel.

6.22.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

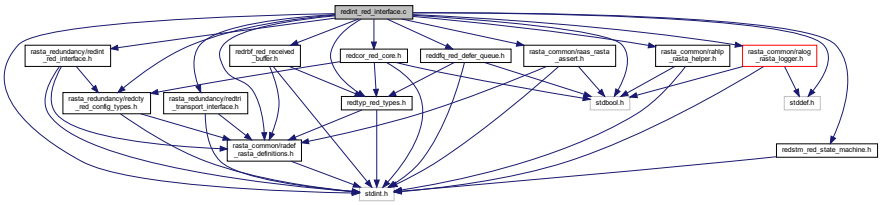
Changelog **-:** Initial version (-, -)

6.23 redint_red_interface.c File Reference

Implementation of RaSTA redundancy layer interface.

```
#include "rasta_redundancy/redint_red_interface.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_redundancy/redcty_red_config_types.h"
#include "rasta_redundancy/redtri_transport_interface.h"
#include "redcor_red_core.h"
#include "reddfq_red_defer_queue.h"
#include "redrbf_red_received_buffer.h"
#include "redstm_red_state_machine.h"
#include "redtyp_red_types.h"
```

Include dependency graph for redint_red_interface.c:



Functions

- static bool IsMessagePending (const uint32_t red_channel_id)

  Returns true, if a received message is pending on any transport channel of a given redundancy channel.
- static void ReceivedMessagesPolling (const uint32_t red_channel_id)

  Polls received messages from the transport channels, copies messages to the input buffer and triggers the state machine for received message processing.
- static void DiscardMessages (const uint32_t red_channel_id)

  Discards all received messages from the transport channels of a given redundancy channel.
- radef_RaStaReturnCode redint_Init (const redcty_RedundancyLayerConfiguration *const redundancy_layer_configuration)

  Initialize the RedL.
- radef_RaStaReturnCode redint_GetInitializationState (void)

  Get the initialization state of the RedL.
- radef_RaStaReturnCode redint_OpenRedundancyChannel (const uint32_t redundancy_channel_id)

  Open a given redundancy channel.
- radef_RaStaReturnCode redint_CloseRedundancyChannel (const uint32_t redundancy_channel_id)

  Close a given redundancy channel.
- radef_RaStaReturnCode redint_SendMessage (const uint32_t redundancy_channel_id, const uint16_t message_size, const uint8_t *const message_data)

  Send a message over a given redundancy channel.
- radef_RaStaReturnCode redint_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 given redundancy channel.
- radef_RaStaReturnCode redint_CheckTimings (void)

  Check redundancy layer timings and read pending messages form the transport channels.

Variables

- PRIVATE radef_RaStaReturnCode **redint_initialization_state** = radef_kNotInitialized

  Initialization state of this module.
- PRIVATE const redcty_RedundancyLayerConfiguration * **redint_redundancy_configuration** = NULL

  Pointer to redundancy layer configuration.
- PRIVATE uint16_t **redint_logger_id**

  ID of the redundancy core debug logger.
- PRIVATE uint32_t tr_channel_polling_read_indices [RADEF_MAX_NUMBER_OF_RED_CHANNELS]

  Transport channel indices for received message polling.

6.23.1 Detailed Description

Implementation of RaSTA redundancy layer interface.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

6.23.2 Function Documentation

6.23.2.1 DiscardMessages() `static void DiscardMessages ( const uint32_t red_channel_id ) [static]`

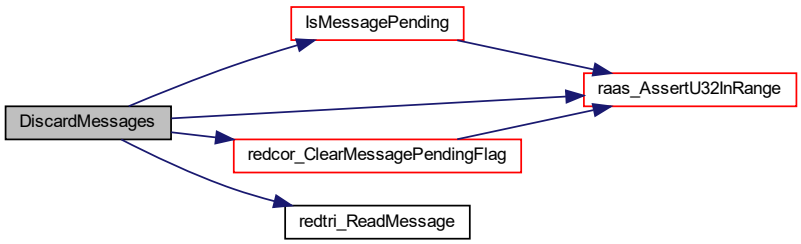
Discards all received messages from the transport channels of a given redundancy channel.

This function discards all received messages from the transport channels of a given redundancy channel. While the received message pending flag of a transport channel is set, the messages are read from this transport channel and they are discarded. If all messages of a transport channel are read, the received messages pending flag of this transport channel is cleared and the reading of messages is started on the next configured transport channel of the given redundancy channel, until all messages of all associated transport channels are read and discarded. This function is used to discard all received messages of a redundancy channel, while this redundancy channel is in closed state.

Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:

6.23.2.2 IsMessagePending() `static bool IsMessagePending ( const uint32_t red_channel_id ) [static]`

Returns true, if a received message is pending on any transport channel of a given redundancy channel.

This function reads and evaluates the received message pending flags of all transport channels associated to the given redundancy channel and returns true, if a received message is pending on any of these transport channels.

Parameters

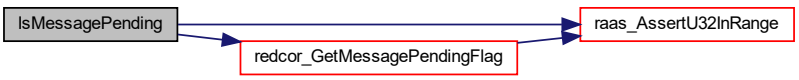
in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Returns

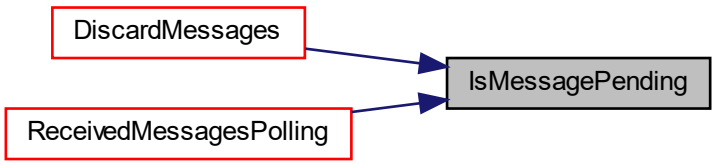
true, if a message is pending on this redundancy channel

false, if no message is pending on this redundancy channel

Here is the call graph for this function:



Here is the caller graph for this function:



6.23.2.3 ReceivedMessagesPolling() `static void ReceivedMessagesPolling ( const uint32_t red_channel_id ) [static]`

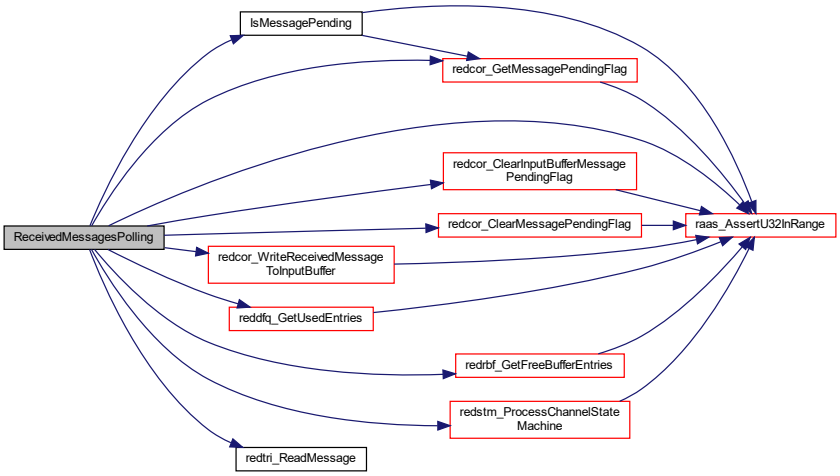
Polls received messages from the transport channels, copies messages to the input buffer and triggers the state machine for received message processing.

This function polls messages from the transport channels. If the received message pending flag is set and the number of free received buffer entries is greater than the number of used defer queue entries, the message is read from the transport channel and copied to the input buffer. Afterwards the state machine is triggered for received message processing. After a polling interruption due to low received buffer capacity, the received message polling continues on the next transport channel. This prevents from restarting the received message polling always with the first transport channel, because this could lead to never reading messages form the other transport channels.

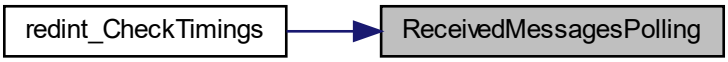
Parameters

in	*red_channel_id*	Redundancy channel identification. Valid range: 0 <= value < configured number of channels.

Here is the call graph for this function:



Here is the caller graph for this function:



6.23.3 Variable Documentation

6.23.3.1 tr_channel_polling_read_indices `PRIVATE uint32_t tr_channel_polling_read_indices[RADEF_MAX_NUMBER_OF_R`

Transport channel indices for received message polling.

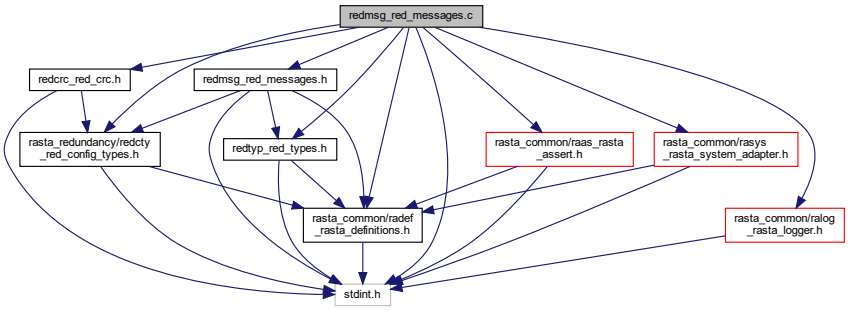
tr_channel_polling_read_indices is used to make sure, that the received message polling continues on the next transport channel, after a polling interruption due to low received buffer capacity. It prevents from restarting the received message polling always with the first transport channel, because this could lead to never reading messages form the other transport channels.

6.24 redmsg_red_messages.c File Reference

Implementation of RaSTA redundancy layer messages module.

```
#include "redmsg_red_messages.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_redundancy/redcty_red_config_types.h"
#include "redcrc_red_crc.h"
#include "redtyp_red_types.h"
```

Include dependency graph for redmsg_red_messages.c:



Enumerations

- enum { kCheckCodeNoneLength = 0U , kCheckCodeCrc16Length = 2U , kCheckCodeCrc32Length = 4U }

  Enum for the check code length.

Functions

- static void SetUint16InMessage (const uint16_t position, const uint16_t data, redtyp_RedundancyMessage *const red_message)

  Set a Uint16 at a specific position in a message.
- static void SetUint32InMessage (const uint16_t position, const uint32_t data, redtyp_RedundancyMessage *const red_message)

  Set a Uint32 at a specific position in a message.
- static void SetPayloadDataInMessage (const redtyp_RedundancyMessagePayload *const message_↩ payload, redtyp_RedundancyMessage *const red_message)

  Set the payload data in a message.
- static uint16_t GetUint16FromMessage (const redtyp_RedundancyMessage *const red_message, const uint16_t position)

  Get a Uint16 from a specific position in a message.
- static uint32_t GetUint32FromMessage (const redtyp_RedundancyMessage *const red_message, const uint16_t position)

  Get a Uint32 from a specific position in a message.
- static uint16_t GetCheckCodeLength (const redcty_CheckCodeType check_code_type)

  Get the length of the configured check code.

- void redmsg_Init (const redcty_CheckCodeType configured_check_code_type)
  Initialize RedL messages module.
- void redmsg_CreateMessage (const uint32_t sequence_number, const redtyp_RedundancyMessagePayload *const message_payload, redtyp_RedundancyMessage *const redundancy_message)
  Create a new redundancy layer message and calculate the check code.
- radef_RaStaReturnCode redmsg_CheckMessageCrc (const redtyp_RedundancyMessage *const redundancy_message)
  Check the check code of a redundancy layer message.
- uint32_t redmsg_GetMessageSequenceNumber (const redtyp_RedundancyMessage *const redundancy_message)
  Get the sequence number of a redundancy layer message.
- void redmsg_GetMessagePayload (const redtyp_RedundancyMessage *const redundancy_message, redtyp_RedundancyMessagePayload *const message_payload)
  Get the payload of a redundancy message.

Variables

- PRIVATE bool **redmsg_initialized** = false
  Initialization state of the module. True, if the module is initialized.
- PRIVATE redcty_CheckCodeType **redmsg_check_code_type**
  Configured check code type.
- static const uint16_t **kByteCountUint16** = 2U
  Byte count of type UInt16_t [bytes].
- static const uint16_t **kByteCountUint32** = 4U
  Byte count of type UInt32_t [bytes].
- static const uint16_t **kMsgLengthPosition** = 0U
  Start position for message length in PDU message [bytes].
- static const uint16_t **kMsgReservePosition** = 2U
  Start position for reserve bytes in PDU message [bytes].
- static const uint16_t **kMsgSequenceNbrPosition** = 4U
  Start position for sequence number in PDU message [bytes].
- static const uint16_t **kMsgPayloadDataPosition** = 8U
  Start position for payload data in PDU message [bytes].
- static const uint16_t **kMsgReserveDataInitValue** = 0U
  Reserve data initial value.

6.24.1 Detailed Description

Implementation of RaSTA redundancy layer messages module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

6.24.2 Enumeration Type Documentation

6.24.2.1 anonymous enum `anonymous enum`

Enum for the check code length.

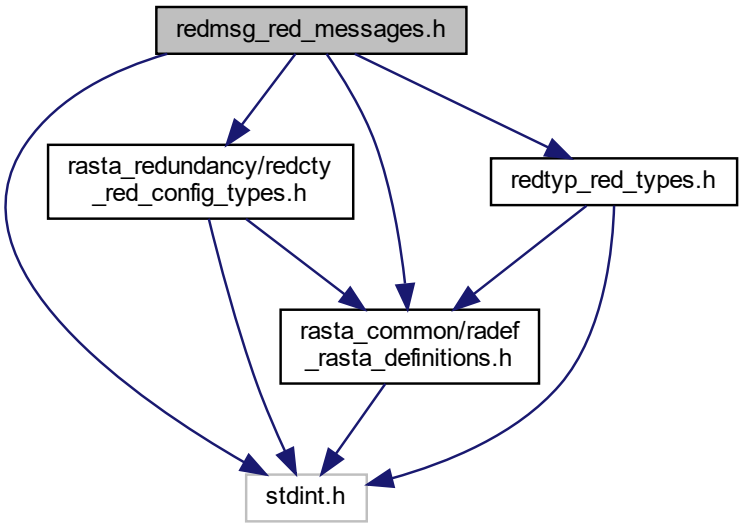
Enumerator

kCheckCodeNoneLength	Length of code code type none [bytes].
kCheckCodeCrc16Length	Length of code code type CRC16 [bytes].
kCheckCodeCrc32Length	Length of code code type CRC32 [bytes].

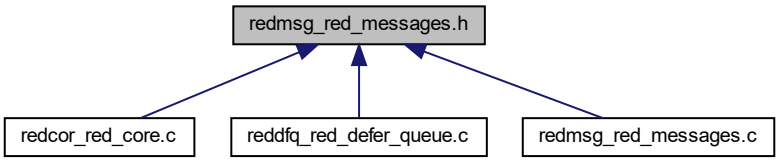
6.25 redmsg_red_messages.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_redundancy/redcty_red_config_types.h"
#include "redtyp_red_types.h"
```

Include dependency graph for redmsg_red_messages.h:



This graph shows which files directly or indirectly include this file:



Functions

- void redmsg_Init (const redcty_CheckCodeType configured_check_code_type)

  Initialize RedL messages module.
- void redmsg_CreateMessage (const uint32_t sequence_number, const redtyp_RedundancyMessagePayload ∗const message_payload, redtyp_RedundancyMessage ∗const redundancy_message)

  Create a new redundancy layer message and calculate the check code.
- radef_RaStaReturnCode redmsg_CheckMessageCrc (const redtyp_RedundancyMessage ∗const redundancy↩ _message)

  Check the check code of a redundancy layer message.
- uint32_t redmsg_GetMessageSequenceNumber (const redtyp_RedundancyMessage ∗const redundancy↩ _message)

  Get the sequence number of a redundancy layer message.
- void redmsg_GetMessagePayload (const redtyp_RedundancyMessage ∗const redundancy_message, redtyp_RedundancyMessagePayload ∗const message_payload)

  Get the payload of a redundancy message.

6.25.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

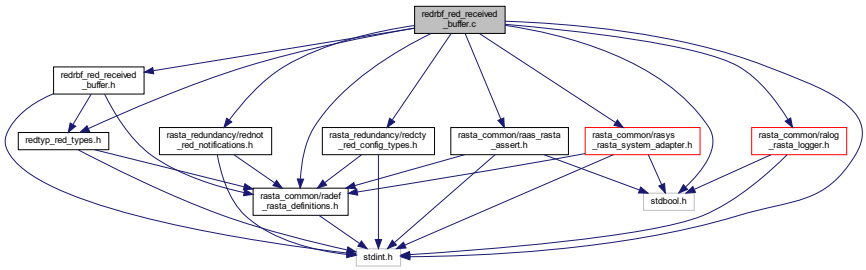
Changelog **-:** Initial version (-, -)

6.26 redrbf_red_received_buffer.c File Reference

Implementation of RaSTA redundancy layer received messages buffer module.

```
#include "redrbf_red_received_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_redundancy/redcty_red_config_types.h"
#include "rasta_redundancy/rednot_red_notifications.h"
#include "redtyp_red_types.h"
```

Include dependency graph for redrbf_red_received_buffer.c:



Classes

- struct ReceivedBuffer

  Struct for redundancy layer received messages payload buffer.

Functions

- void redrbf_Init (const uint32_t configured_red_channels)

  Initialize the RedL received buffer module.
- void redrbf_InitBuffer (const uint32_t red_channel_id)

  Initialize the received buffer of a dedicated redundancy channel.
- void redrbf_AddToBuffer (const uint32_t red_channel_id, const redtyp_RedundancyMessagePayload ∗const message_payload)

  Add a RedL message to the received buffer of a dedicated redundancy channel. A fatal error is raised, if the buffer is full.
- radef_RaStaReturnCode redrbf_ReadFromBuffer (const uint32_t red_channel_id, const uint16_t buffer_size, uint16_t ∗const message_size, uint8_t ∗const message_buffer)

  Read and remove a RedL message payload from the received buffer of a dedicated redundancy channel.
- uint16_t redrbf_GetFreeBufferEntries (const uint32_t red_channel_id)

  Get the number of free buffer entries [messages].

Variables

- PRIVATE bool **redrbf_initialized** = false

  Initialization state of the module. True, if the module is initialized.
- PRIVATE uint32_t **redrbf_number_of_red_channels** = 0U

  Number of configured redundancy channels.
- PRIVATE ReceivedBuffer **redrbf_received_buffers** [RADEF_MAX_NUMBER_OF_RED_CHANNELS]

  Received buffers for all redundancy channels.

6.26.1 Detailed Description

Implementation of RaSTA redundancy layer received messages buffer module.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

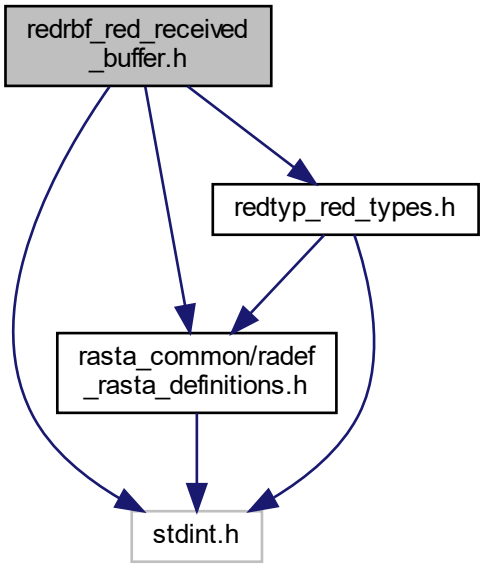
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

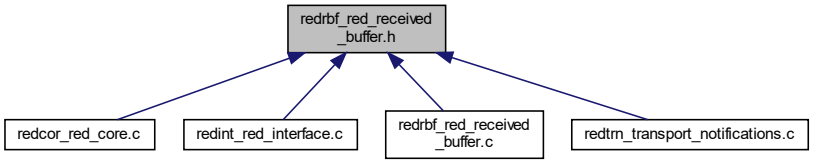
6.27 redrbf_red_received_buffer.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
#include "redtyp_red_types.h"
```

Include dependency graph for redrbf_red_received_buffer.h:



This graph shows which files directly or indirectly include this file:



Functions

- void redrbf_Init (const uint32_t configured_red_channels)

  Initialize the RedL received buffer module.
- void redrbf_InitBuffer (const uint32_t red_channel_id)

  Initialize the received buffer of a dedicated redundancy channel.
- void redrbf_AddToBuffer (const uint32_t red_channel_id, const redtyp_RedundancyMessagePayload ∗const message_payload)

  Add a RedL message to the received buffer of a dedicated redundancy channel. A fatal error is raised, if the buffer is full.
- radef_RaStaReturnCode redrbf_ReadFromBuffer (const uint32_t red_channel_id, const uint16_t buffer_size, uint16_t ∗const message_size, uint8_t ∗const message_buffer)

  Read and remove a RedL message payload from the received buffer of a dedicated redundancy channel.
- uint16_t redrbf_GetFreeBufferEntries (const uint32_t red_channel_id)

  Get the number of free buffer entries [messages].

6.27.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

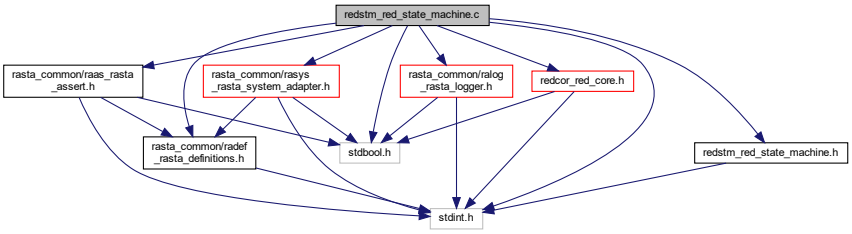
Changelog **-:** Initial version (-, -)

6.28 redstm_red_state_machine.c File Reference

Implementation of RaSTA redundancy layer state machine.

```
#include "redstm_red_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 "redcor_red_core.h"
```

Include dependency graph for redstm_red_state_machine.c:



Functions

- static void ProcessStateClosedEvents (const uint32_t red_channel_id, const redstm_RedundancyChannelEvents event)

  Process events in the closed state.
- static void ProcessStateUpEvents (const uint32_t red_channel_id, const redstm_RedundancyChannelEvents event)

  Process events in the up state.
- void redstm_Init (const uint32_t configured_red_channels)

  Initialize the RedL state machine module.
- void redstm_ProcessChannelStateMachine (const uint32_t red_channel_id, const redstm_RedundancyChannelEvents event)

  Process redundancy channel state machine.
- redstm_RedundancyChannelStates redstm_GetChannelState (const uint32_t red_channel_id)

  Return the state of a redundancy channel state machine.

Variables

- PRIVATE bool **redstm_initialized** = false

  Initialization state of the module. True, if the module is initialized.
- PRIVATE uint32_t **redstm_number_of_red_channels** = 0U

  Number of configured redundancy channels.
- PRIVATE redstm_RedundancyChannelStates redstm_redundancy_channel_states [RADEF_MAX_NUMBER_OF_RED_CHAN

  States of the redundancy channel state machines.

6.28.1 Detailed Description

Implementation of RaSTA redundancy layer state machine.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

6.28.2 Variable Documentation

6.28.2.1 redstm_redundancy_channel_states

```
PRIVATE redstm_RedundancyChannelStates redstm_↩
redundancy_channel_states[RADEF_MAX_NUMBER_OF_RED_CHANNELS]
```

Initial value:

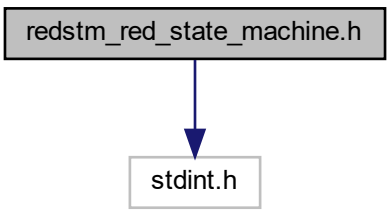
```
= {redstm_kRedundancyChannelStateNotInitialized,

      redstm_kRedundancyChannelStateNotInitialized}
```

States of the redundancy channel state machines.

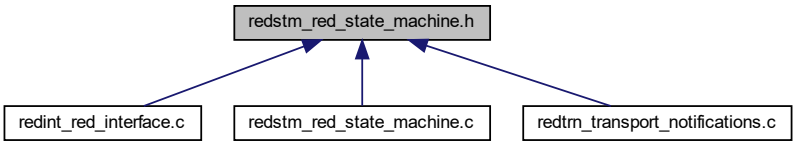
6.29 redstm_red_state_machine.h File Reference

```
#include <stdint.h>
```

Include dependency graph for redstm_red_state_machine.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum redstm_RedundancyChannelStates { redstm_kRedundancyChannelStateMin = 0U , redstm_kRedundancyChannelStateNotInitialized = 0U , redstm_kRedundancyChannelStateClosed , redstm_kRedundancyChannelStateUp , redstm_kRedundancyChannelStateMax }

  Enum for the states of a redundancy channel state machine.
- enum redstm_RedundancyChannelEvents { redstm_kRedundancyChannelEventMin = 0U , redstm_kRedundancyChannelEventOpen = 0U , redstm_kRedundancyChannelE , redstm_kRedundancyChannelEventReceiveData , redstm_kRedundancyChannelEventSendData , redstm_kRedundancyChannelEventDeferTimeout , redstm_kRedundancyChannelEventMax }

  Enum for the events of a redundancy channel state machine.

Functions

- void redstm_Init (const uint32_t configured_red_channels)

  Initialize the RedL state machine module.
- void redstm_ProcessChannelStateMachine (const uint32_t red_channel_id, const redstm_RedundancyChannelEvents event)

  Process redundancy channel state machine.
- redstm_RedundancyChannelStates redstm_GetChannelState (const uint32_t red_channel_id)

  Return the state of a redundancy channel state machine.

6.29.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

6a56b6d9a6998494e007e63d764a40fdf63cbbac

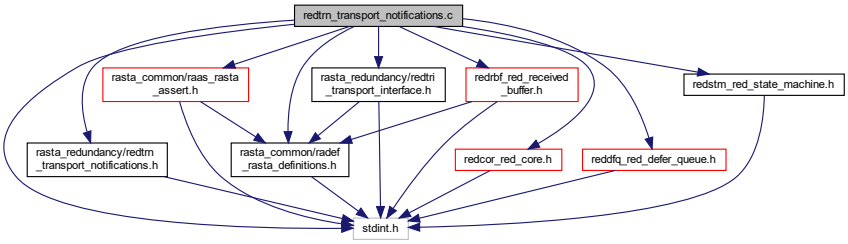
Changelog **-:** Initial version (-, -)

6.30 redtrn_transport_notifications.c File Reference

Implementation of RaSTA transport layer notification functions.

```
#include "rasta_redundancy/redtrn_transport_notifications.h"
#include <stdint.h>
#include "rasta_common/raas_rasta_assert.h"
#include "rasta_common/radef_rasta_definitions.h"
#include "rasta_redundancy/redtri_transport_interface.h"
#include "redcor_red_core.h"
#include "reddfq_red_defer_queue.h"
#include "redrbf_red_received_buffer.h"
#include "redstm_red_state_machine.h"
```

Include dependency graph for redtrn_transport_notifications.c:



Functions

- void redtrn_MessageReceivedNotification (const uint32_t transport_channel_id)

  Transport layer message received notification function.

6.30.1 Detailed Description

Implementation of RaSTA transport layer notification functions.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

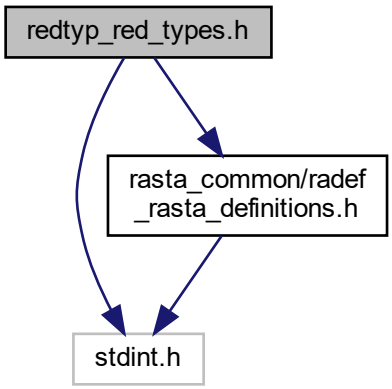
6a56b6d9a6998494e007e63d764a40fdf63cbbac

Changelog **-:** Initial version (-, -)

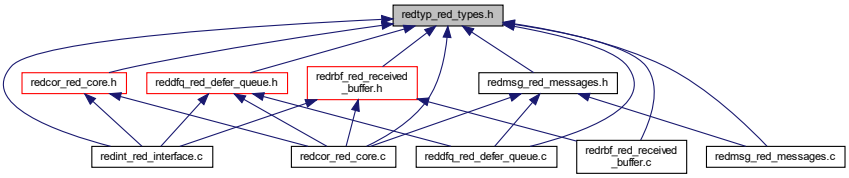
6.31 redtyp_red_types.h File Reference

```
#include <stdint.h>
#include "rasta_common/radef_rasta_definitions.h"
```

Include dependency graph for redtyp_red_types.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct redtyp_RedundancyMessage

  Typedef for a redundancy layer PDU message.
- struct redtyp_RedundancyMessagePayload

  Typedef for a redundancy layer PDU message payload.

6.31.1 Detailed Description

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

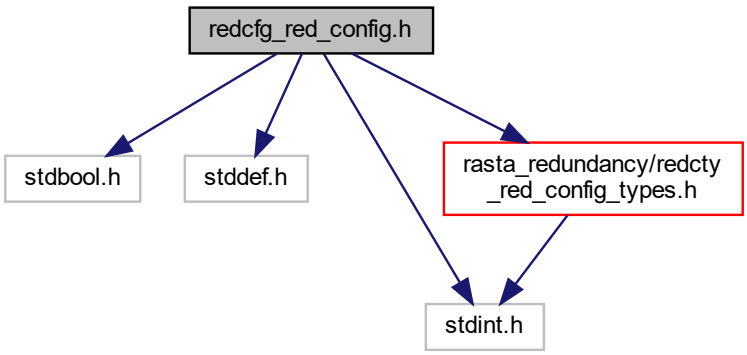
6a56b6d9a6998494e007e63d764a40fdf63cbbac

6.32 redcfg_red_config.h File Reference

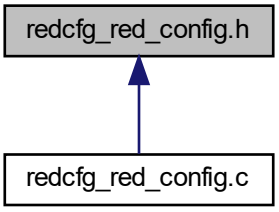
Interface of RaSTA redundancy layer configuration.

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_redundancy/redcty_red_config_types.h"
```

Include dependency graph for redcfg_red_config.h:



This graph shows which files directly or indirectly include this file:



Variables

- const redcty_RedundancyLayerConfiguration **redundancy_layer_configuration**
  Configuration data of a redundancy layer.

6.32.1 Detailed Description

Interface of RaSTA redundancy layer configuration.

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

cb4315a92d98c9390dbd583ad9cdd788738cc54c

Changelog **-:** Initial version (-, -)

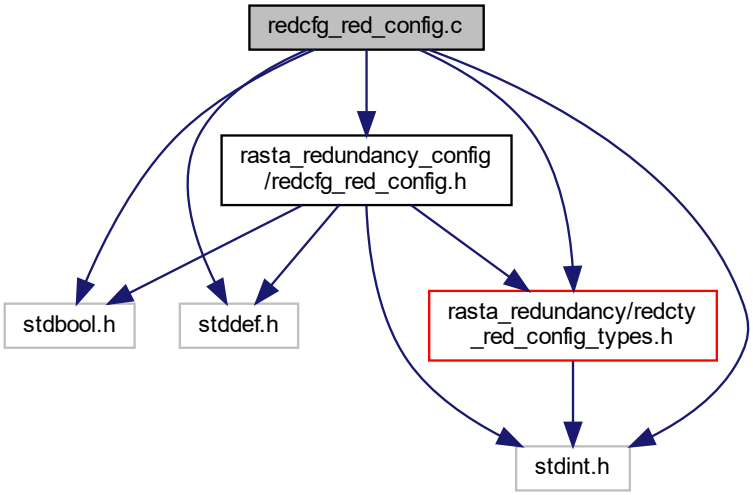
This module provides the configuration data structure for the RedL.

6.33 redcfg_red_config.c File Reference

Definition of RaSTA redundancy layer configuration. Detailed description of this library ...

```
#include "rasta_redundancy_config/redcfg_red_config.h"
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include "rasta_redundancy/redcty_red_config_types.h"
```

Include dependency graph for redcfg_red_config.c:



Variables

- const redcty_RedundancyLayerConfiguration redundancy_layer_configuration

  Configuration data of a redundancy layer.

6.33.1 Detailed Description

Definition of RaSTA redundancy layer configuration. Detailed description of this library ...

Copyright

Copyright (C) 2022, SBB AG, CH-3000 Bern

Author

Roland Schenk, CSA Engineering AG, CH-4500 Solothurn

Version

cb4315a92d98c9390dbd583ad9cdd788738cc54c

Changelog **-:** Initial version (-, -)

6.33.2 Variable Documentation

6.33.2.1 redundancy_layer_configuration `const redcty_RedundancyLayerConfiguration redundancy_↩ layer_configuration`

Initial value:

```
= {
    .check_code_type = redcty_kCheckCodeA,
    .t_seq = 50U,
    .n_diagnosis = 200U,
    .n_defer_queue_size = 4U,
    .number_of_redundancy_channels = 2U,
    {
        {
            .red_channel_id = 0U,
            .num_transport_channels = 2U,
            .transport_channel_ids = {0U, 1U},
        },
        {
            .red_channel_id = 1U,
            .num_transport_channels = 2U,
            .transport_channel_ids = {2U, 3U},
        },
    },
}
```

Configuration data of a redundancy layer.

Index