Document Title	Specification of Raw Data Stream
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	1098

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R23-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2023-11-23	R23-11	AUTOSAR Release Management	• Initial release

Disclaimer

Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the functional cluster `RAW Data Stream` for the AUTOSAR Adaptive Platform.

In some cases it is necessary for the application software to be able to process raw binary data streams sent over a communication channel. In a raw binary data stream the data is not typed, and is handled as a continuing sequence of bytes. So serialization of the data is not necessary. This functional cluster specifies an interface to support processing of raw binary data streams.

The interface is statically defined and independent of the underlying network protocol. However, currently the modeling for the Raw Data Streaming Interface only supports TCP/IP sockets as transport layer. Both unicast and multicast socket connections shall be supported. The sockets can use both TCP or UDP as transport protocol. TCP is the natural choice for RawDataStreams since it is a reliable stream oriented protocol. However, UDP shall also be supported when an unreliable connection is acceptable for the application.

The integration of the Raw Data Streaming Interface and Adaptive Applications is done in the deployment phase, by specifying various attributes and parameters for the socket connections that shall be used for the Raw Data Stream.

Secure communication can be achieved by applying TLS or IPSec protocols in the middleware. Also access control imposed by the IAM can be applied for Raw Data Streams.

For safety critical applications wanting to use RawDataStreaming, a safety analysis needs to be done by the application developer, to find relevant communication faults for the stream data. If a protection of data exchange algorithm is needed, such as E2E protection, this will not be provided in the RawDataStream interface, but is to be implemented in the application layer that is using the RawDataStream interface. This is because only raw data with no data type information is transferred over the RawDataStream.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the `RAW Data Stream` module that are not included in the [1, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
IP	Internet Protocol
SOME/IP	Scalable service-Oriented MiddlewarE over IP
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
E2E	End-to-end communication protection
TLS	Transport Layer Security
DTLS	Datagram Transport Layer Security

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General

[3] Requirements on Communication Management
AUTOSAR_AP_RS_CommunicationManagement

[4] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[5] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[6] Specification of Communication Management
AUTOSAR_AP_SWS_CommunicationManagement

3.2 Further applicable specification

AUTOSAR provides a general specification [2, RS General] which is also applicable for `RAW Data Stream`. The specification SWS General shall be considered as additional and required specification for implementation of `RAW Data Stream`.

Currently, the specific requirements for `RAW Data Stream` are part of [3, RS Communication Management].

4 Constraints and assumptions

4.1 Known limitations

The current solution does not support any runtime variance in terms of network topology, such as service discovery functionality, which means that the RawDataStreams has to be configured statically on the same ECU as the application. Dynamic configuration and runtime functionality will be added in future releases if needed.

The multicast support is limited to one-to-many, i.e. a server can send data to multiple clients using multicast, but only receive data from one client, using the unicast address. Also multicast shall only be used with UDP. For TCP connections, only 1-to-1 connections are supported, i.e. multiple clients to one server is not supported.

5 Dependencies to other Functional Clusters

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 “Provided Interfaces” lists the interfaces provided by `Raw Data Stream` to other Functional Clusters. Section 5.2 “Required Interfaces” lists the interfaces required by `Raw Data Stream`.

A detailed technical architecture documentation of the AUTOSAR Adaptive Platform is provided in [4].

5.1 Provided Interfaces

Table 5.1 provides a complete list of interfaces provided to other Functional Clusters within the AUTOSAR Adaptive Platform.

Interface	*Functional Cluster*	*Purpose*
No provided interfaces		

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

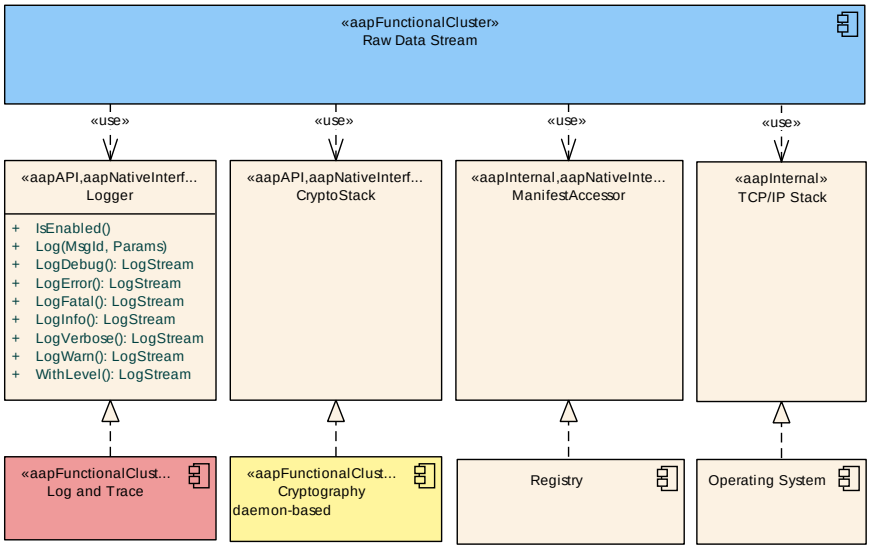

Figure 5.1: Interfaces required by Raw Data Stream from other Functional Clusters

Figure 5.1 shows interfaces required by `Raw Data Stream` from other Functional Clusters within the AUTOSAR Adaptive Platform. Table 5.2 provides a complete list of required interfaces from other Functional Clusters within the AUTOSAR Adaptive Platform.

Functional Cluster	*Interface*	*Purpose*
Cryptography	CryptoStack	This interface may be used to establish encrypted connections.
Log and Trace	Logger	`Raw Data Stream` uses this interface to log standardized messages.

Table 5.2: Interfaces required from other Functional Clusters

6 Requirements Tracing

The following tables reference the requirements specified in the Requirements on Communication Management document [3] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_AP_00120]	Method and Function names.	[SWS_RDS_11292] [SWS_RDS_11294] [SWS_RDS_11295] [SWS_RDS_11296] [SWS_RDS_11297] [SWS_RDS_11298] [SWS_RDS_11299]
[RS_AP_00121]	Parameter names.	[SWS_RDS_11292] [SWS_RDS_11296] [SWS_RDS_11297] [SWS_RDS_11299]
[RS_AP_00122]	Type names.	[SWS_RDS_11291] [SWS_RDS_11293] [SWS_RDS_12367]
[RS_AP_00127]	Usage of ara::core types.	[SWS_RDS_11291] [SWS_RDS_11293] [SWS_RDS_12367]
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment.	[SWS_RDS_11291] [SWS_RDS_11292] [SWS_RDS_11293] [SWS_RDS_11294] [SWS_RDS_11295] [SWS_RDS_11296] [SWS_RDS_11297] [SWS_RDS_11298] [SWS_RDS_11299] [SWS_RDS_12367] [SWS_RDS_99025]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_RDS_11292] [SWS_RDS_11295] [SWS_RDS_11296] [SWS_RDS_11298] [SWS_RDS_11299]
[RS_AP_00145]	Availability of special member functions.	[SWS_RDS_10482] [SWS_RDS_10483] [SWS_RDS_11303] [SWS_RDS_11304] [SWS_RDS_11305] [SWS_RDS_11306] [SWS_RDS_11312] [SWS_RDS_11313] [SWS_RDS_11314] [SWS_RDS_11315] [SWS_RDS_11316] [SWS_RDS_11317]
[RS_AP_00146]	Classes whose construction requires interaction by the ARA framework.	[SWS_RDS_11294]
[RS_AP_00147]	Classes that are created with an InstanceSpecifier as an argument are not copyable, but at most movable.	[SWS_RDS_11303] [SWS_RDS_11304] [SWS_RDS_11305] [SWS_RDS_11306] [SWS_RDS_11316] [SWS_RDS_11317]
[RS_CM_00001]	The Communication Management shall provide a standardized header file structure for each service.	[SWS_RDS_10488] [SWS_RDS_10490]
[RS_CM_00002]	The service header files shall define the namespace for the respective service.	[SWS_RDS_10489]

▽

△

Requirement	Description	Satisfied by
[RS_CM_00410]	The Communication Management shall provide an API to support reading and writing raw data streams that has no datatype information	[SWS_RDS_10476] [SWS_RDS_10477] [SWS_RDS_10478] [SWS_RDS_10479] [SWS_RDS_10480] [SWS_RDS_10481] [SWS_RDS_10482] [SWS_RDS_10483] [SWS_RDS_10484] [SWS_RDS_10485] [SWS_RDS_10486] [SWS_RDS_10487] [SWS_RDS_10508] [SWS_RDS_11300] [SWS_RDS_11301] [SWS_RDS_11302] [SWS_RDS_11303] [SWS_RDS_11304] [SWS_RDS_11305] [SWS_RDS_11306] [SWS_RDS_11307] [SWS_RDS_11309] [SWS_RDS_11310] [SWS_RDS_11311] [SWS_RDS_11312] [SWS_RDS_11313] [SWS_RDS_11314] [SWS_RDS_11315] [SWS_RDS_11316] [SWS_RDS_11317] [SWS_RDS_11318] [SWS_RDS_11319] [SWS_RDS_11320] [SWS_RDS_11322] [SWS_RDS_11323] [SWS_RDS_11324] [SWS_RDS_11325] [SWS_RDS_90216] [SWS_RDS_90217] [SWS_RDS_99004] [SWS_RDS_99006]
[RS_CM_00411]	Application developers shall be able to send and receive raw binary data streams independent of the underlying network protocol	[SWS_RDS_10476] [SWS_RDS_10477] [SWS_RDS_10478] [SWS_RDS_10479] [SWS_RDS_10480] [SWS_RDS_10481] [SWS_RDS_10482] [SWS_RDS_10483] [SWS_RDS_10484] [SWS_RDS_10485] [SWS_RDS_10486] [SWS_RDS_10487] [SWS_RDS_10508] [SWS_RDS_11300] [SWS_RDS_11301] [SWS_RDS_11302] [SWS_RDS_11303] [SWS_RDS_11304] [SWS_RDS_11305] [SWS_RDS_11306] [SWS_RDS_11307] [SWS_RDS_11309] [SWS_RDS_11310] [SWS_RDS_11311] [SWS_RDS_11312] [SWS_RDS_11313] [SWS_RDS_11314] [SWS_RDS_11315] [SWS_RDS_11316] [SWS_RDS_11317] [SWS_RDS_11318] [SWS_RDS_11319] [SWS_RDS_11320] [SWS_RDS_11322] [SWS_RDS_11323] [SWS_RDS_11324] [SWS_RDS_11325] [SWS_RDS_90216] [SWS_RDS_90217] [SWS_RDS_99004] [SWS_RDS_99005] [SWS_RDS_99006]
[RS_CM_00412]	The Communication Management shall provide TCP/IP Sockets as network protocol for Raw Data Streams	[SWS_RDS_10476] [SWS_RDS_10477] [SWS_RDS_10478] [SWS_RDS_10479] [SWS_RDS_10480] [SWS_RDS_10482] [SWS_RDS_10483] [SWS_RDS_10484] [SWS_RDS_10485] [SWS_RDS_10486] [SWS_RDS_10487] [SWS_RDS_10508] [SWS_RDS_11300] [SWS_RDS_11301] [SWS_RDS_11302] [SWS_RDS_11303] [SWS_RDS_11304] [SWS_RDS_11305] [SWS_RDS_11306] [SWS_RDS_11307] [SWS_RDS_11309] [SWS_RDS_11310] [SWS_RDS_11312] [SWS_RDS_11313] [SWS_RDS_11314] [SWS_RDS_11315] [SWS_RDS_11316] [SWS_RDS_11317] [SWS_RDS_11318] [SWS_RDS_11319] [SWS_RDS_11320] [SWS_RDS_11322] [SWS_RDS_11323] [SWS_RDS_11324] [SWS_RDS_11325] [SWS_RDS_90216] [SWS_RDS_99004] [SWS_RDS_99005]

▽

△

Requirement	Description	Satisfied by
[RS_CM_00801]	Secure communication shall be transmitted using secure channels	[SWS_RDS_90211] [SWS_RDS_90212] [SWS_RDS_90213] [SWS_RDS_90214] [SWS_RDS_90215]
[RS_CM_00803]	The assignment of communication to specific secure channels shall be configurable	[SWS_RDS_90212]
[RS_IAM_00006]	Access control policies shall be available to the PDP	[SWS_RDS_90007]
[RS_IAM_00007]	The `Adaptive Platform Foundation` shall provide access control decisions	[SWS_RDS_90007]
[RS_IAM_00010]	Adaptive applications shall only be able to use AUTOSAR Resources when authorized	[SWS_RDS_90007]

Table 6.1: RequirementsTracing

7 Functional specification

7.1 Raw Data Streaming Interface

The operations of the interface are synchronous. The default behavior is blocking, but a timeout handling shall be implemented to return the call with an error if the operation takes too long. The timeout values are applied as parameters to each operation. See the description for each operation below on how the timeout handling is applied.

The configuration of the Raw Data Streams is done by specifying credentials and parameters for the socket connections that shall be used for the Raw Data Stream, using `RawDataStreamMapping` and `EthernetRawDataStreamMapping`. The model elements and the parameters are described in *TPS_ManifestSpecification* [5].

Secure communication can be achieved by applying TLS or IPSec protocols in the middleware. Also access control imposed by the IAM can be applied for Raw Data Streams. All security functions are configurable in the deployment and mapping model of Raw Data Streaming Interface, see *TPS_ManifestSpecification* [5].

All security functions (TLS, IPSec, IAM) are configurable in the deployment and mapping model of Raw Data Streaming Interface, see *TPS_ManifestSpecification* [5].

An application can use the Raw Data Streaming API both as a client (connecting to a listening Raw Data Streaming service) or server (waiting for incoming connections from clients).

Figure 7.1 shows the logical view of the usage of RawDataStream instances.

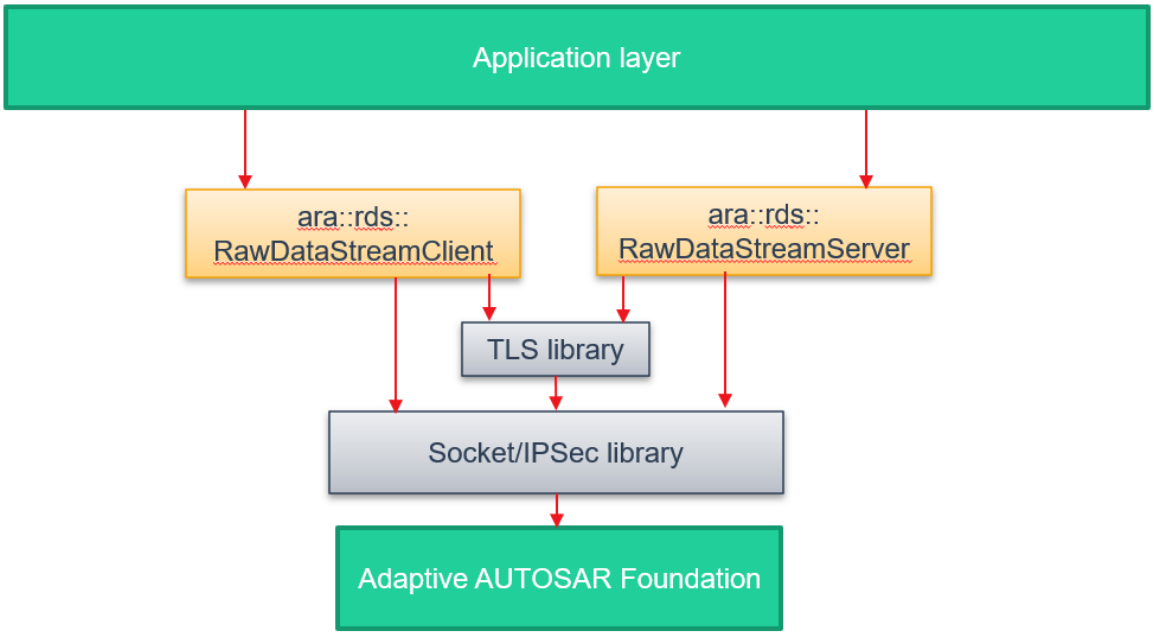

Figure 7.1: Raw Data Stream Logical View.

7.1.1 Use cases

The RawDataStream interface can be used in the following set-ups:

- Client (connect to) to an external non-AUTOSAR sensor providing raw data on a socket connection.
- Server (wait for a connection from) for an external non-AUTOSAR sensor providing raw data on a socket connection.
- Client or Server for another AUTOSAR external RawDataStream instance.

RawDataStream socket connections can be setup for UDP or TCP, Unicast or Multicast. Currently the use cases in fig 7.2 are supported.

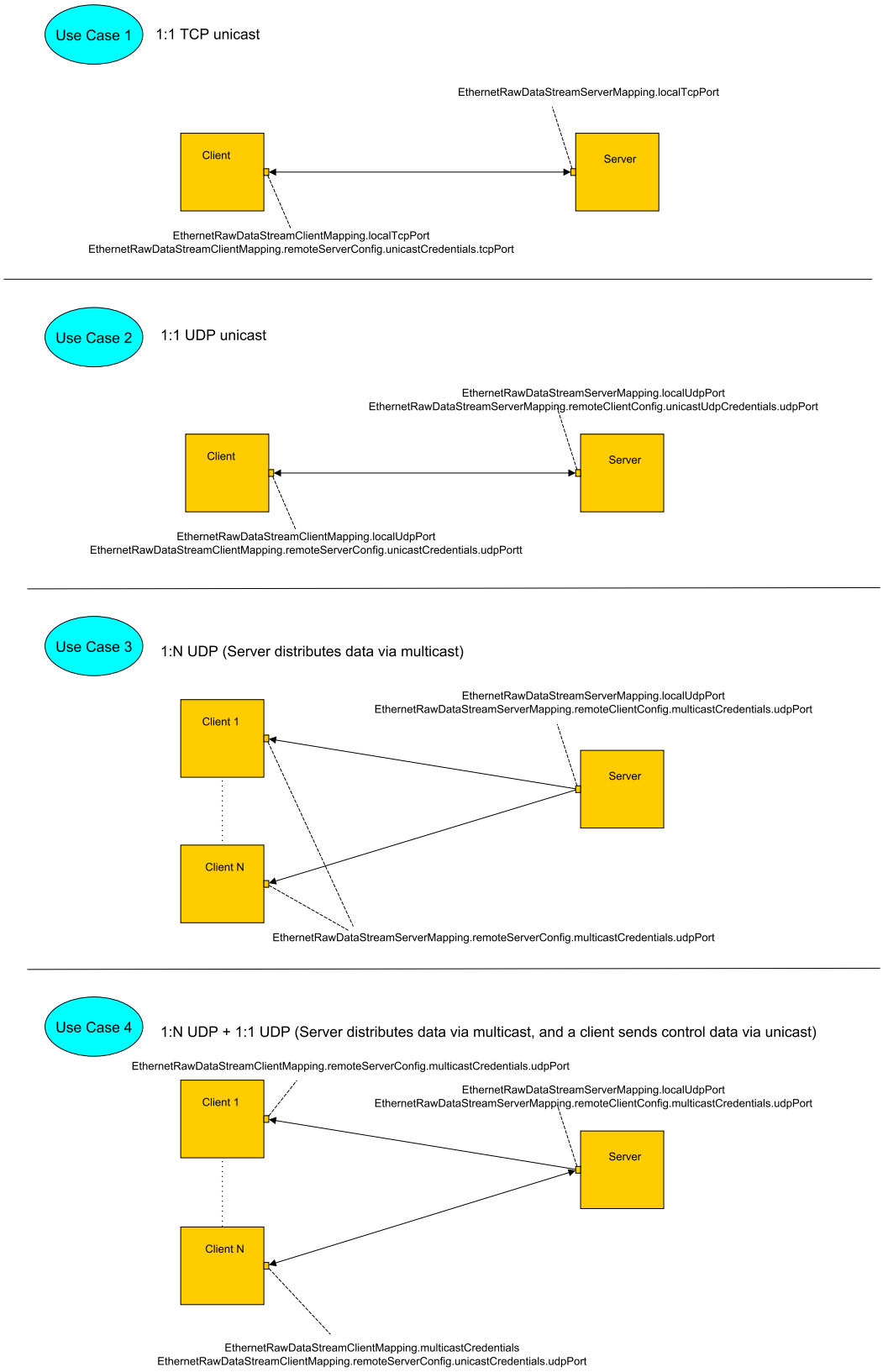

Figure 7.2: The currently supported use cases for Raw Data Streams, and which artifacts in the Deployment model that shall be used to configure the different use cases

7.2 Raw Data Streaming

For the Raw Data Stream C++ API reference, see chapter 8.

[SWS_RDS_10476] Defining a RawDataStream ⌈To open a `RawDataStream` connection a `RawDataStream` instance is created. The constructor creates the necessary socket data structures for `RawDataStream` Communication, using the artifacts specified in the mapped `EthernetRawDataStreamClientMapping` and `EthernetRawDataStreamServerMapping`.⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_RDS_99004]{DRAFT} **Ethernet endpoint configuration** ⌈

Ethernet socket connections are statically configured in the Deployment model as part of the Service Instance Manifest, and used throughout the connected session for the RawDataStreams communication. The following configuration elements can be specified on the Deployment model of each RawDataStreamClient or RawDataStreamServer instance, identified through the InstanceSpecifier provided to the constructor.

RawDataStreamClient endpoint and credentials configuration elements:

- Local Network Endpoint: `EthernetRawDataStreamClientMapping.localCommConnector`
- Local UdpPort: `EthernetRawDataStreamClientMapping.localUdpPort`
- Local TcpPort: `EthernetRawDataStreamClientMapping.localTcpPort`
- Socket Options: `EthernetRawDataStreamClientMapping.socketOption`
- (D)TLS properties: `EthernetRawDataStreamClientMapping.tlsSecureComProps`
- Remote Unicast Credentials: `EthernetRawDataStreamClientMapping.unicastCredentials` (UDP/TCP)
- Multicast Credentials: `EthernetRawDataStreamClientMapping.multicastCredentials` (UDP only)

RawDataStreamServer endpoint and credentials configuration elements:

- Local Network Endpoint: `EthernetRawDataStreamServerMapping.localCommConnector`
- Local UdpPort: `EthernetRawDataStreamServerMapping.localUdpPort`
- Local TcpPort: `EthernetRawDataStreamServerMapping.localTcpPort`
- Socket Options: `EthernetRawDataStreamServerMapping.socketOption`
- (D)TLS properties: `EthernetRawDataStreamServerMapping.tlsSecureComProps`
- Remote Unicast Credentials: `EthernetRawDataStreamServerMapping.unicastUdpCredentials` (UDP only)

- Multicast Credentials: EthernetRawDataStreamServerMapping.multicastCredentials (UDP only)

For the RawDataStreamClients the following shall apply:

- Remote server credentials for unicast communication must always be defined for the client. The Unicast remote server credentials are configured in RawDataStreamEthernetTcpUdpCredentials aggregated by the EthernetRawDataStreamClientMapping in the role unicastCredentials.
- A tcpPort and udpPort shall not be defined in the same RawDataStreamEthernetTcpUdpCredentials element.
- If a TcpPort is defined in the EthernetRawDataStreamClientMapping.unicastCredentials, these credentials are used for Connect() calls to establish the connection to the server.
- This unicast connection shall always be used for WriteData() calls to send data to the server (for both UDP and TCP).
- If Multicast Credentials are defined for the client, the RawDataStream shall bind and join the multicast address and udpPort given in the MulticastCredentials. The MulticastCredentials is configured in RawDataStreamEthernetUdpCredentials aggregated by the EthernetRawDataStreamClientMapping. This multicast socket connection shall be read from when ReadData() is called.
- If no MulticastCredentials are defined for the client, the Unicast Remote Credentials shall also be used for ReadData() calls.

For the RawDataStreamServers the following shall apply:

- If Multicast Credentials is defined for the server, a multicast connection shall be created using the Multicast Credentials which are configured in RawDataStreamEthernetUdpCredentials aggregated by the EthernetRawDataStreamServerMapping in the role multicastCredentials. Then the data is sent on this multicast socket when WriteData() is called.
- If Remote Unicast Credentials are defined for the server, a unicast socket shall be created using the Unicast Credentials which are configured in RawDataStreamEthernetUdpCredentials aggregated by the EthernetRawDataStreamServerMapping in the role unicastUdpCredentials. Then the data is sent on this unicast socket when WriteData() is called.
- The local credentials defined in EthernetCommunicationConnector shall always be used to create a unicast socket and read data from a client when ReadData() is called on the server side. If no local credentials are defined, reading of data from the server cannot be performed, and an error kStreamNotConnected will be returned.
- If a localTcpPort is defined in EthernetRawDataStreamServerMapping, the credentials defined in EthernetCommunicationConnector are used to create, bind, and listen to the socket used for TCP communication when the

constructor of RawDataStream is called. Then the server accepts incoming connection requests when WaitForConnection() is called.

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_90216]{DRAFT} **Socket Options configuration** ⌈For both RawDataStreamClients and RawDataStreamServers a list of socket options can be defined in the attribute `socketOption` to be applied to the sockets created for unicast or multicast communication. The options shall be specified as a list of strings. The accepted values are platform specific and shall be documented by the vendor.⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

An example of `socketOption` definition is to provide a series of "option", "value" pairs for POSIX socket level options, e.g.: ["SO_KEEPALIVE", "1", SO_RCVBUF","1024"]

[SWS_RDS_90217]{DRAFT} **TLS properties configuration** ⌈For both RawDataStreamClients and RawDataStreamServers (D)TLS properties can be defined in the attributes `tlsSecureComProps` to configure usage of TLS to create secure UDP and TCP channels for the RawDataStreams according to the Transport Layer Security protocol. See [SWS_RDS_90211]⌋(*RS_CM_00410*, *RS_CM_00411*)

Note: Usage of (D)TLS is restricted to 1:1 socket connections (use case 1 and 2 of figure Figure 7.2).

The functionality of a `RawDataStream` for Client communication is realized in these four operations: Connect, Shutdown, ReadData and WriteData. A `RawDataStream` for Server Communication is realized in these four operations: WaitForConnection, Shutdown, ReadData and WriteData.

[SWS_RDS_10477] Connect stream link ⌈Each invocation of the Connect operation for a TCP socket connection shall establish a communication link with a remote server that is listening for socket connections, The socket created in the `RawDataStream` instance shall be used for the connection. For UDP socket connections Connect shall do nothing.⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_99005]{DRAFT} **Wait for incoming connections** ⌈Each invocation of the WaitForConnection operation shall wait for and accept incoming requests for establishment of a TCP communication link with a connecting remote client. The socket created and prepared in the RawDataStream instance shall be used for the connection. For UDP socket connections WaitForConnection shall do nothing.⌋(*RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_10478] Shutdown stream link ⌈Each invocation of the Shutdown operation shall destroy the communication link for the stream.⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_10508] Destructor behavior when Shutdown stream link ⌈If the destructor is executed on an instance on which no Shutdown operation has been performed, the destructor shall perform Shutdown internally before the object is destroyed.⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_10479] Read data from stream ⌈Each invocation of the ReadData operation shall request to read a number of bytes from the stream. The read data shall be moved to a buffer returned as result from the function, together with the actual number of bytes transferred.⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_RDS_10480] Write data to stream ⌈Each invocation of the WriteData operation shall request to write a number of bytes to the stream and send it out on the socket connection. The actual number of bytes transferred shall be returned. It shall be possible to apply a timeout value for the operation. The operation shall write the data to the socket or internal buffer, and then return with the number of bytes written. For efficiency, the Write operation does not wait until data is actually sent on the bus, but the TCP data flow handling shall make sure that data is transmitted and received in the correct order. For UDP connections the order cannot be guaranteed.⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_RDS_99006]{DRAFT} **Timeout handling** ⌈For all Connect, WaitForConnection, Read and Write `RawDataStream` operations a timeout value can be specified via a parameter in runtime. If no timeout parameter is given the operation shall block. If a timeout value is specified, and the operation does not finish within the specified time, an error code `RawErrc::kCommunicationTimeout` shall be returned and the technical state of the `RawDataStream` connection shall be restored to the same as before the call was made.⌋*(RS_CM_00410, RS_CM_00411)*

7.3 Security

7.3.1 Access Control via IAM

[SWS_RDS_90007] Restrictions on using RawDataStreams ⌈If a `Process` calls

- the `Connect()` method of the `RawDataStreamClient` class (see [SWS_RDS_10484])

or

- the `WaitForConnection()` method of the `RawDataStreamServer` class (see [SWS_RDS_11318])

but there exists no `EthernetRawDataStreamGrant` that references the used `EthernetRawDataStreamMapping` in the role `ethernetRawDataStreamMapping` which references the requesting `Process` in the role `process`, then the methods shall return `RawErrc::kGrantEnforcementError` in the `Result` of the method.

⌋*(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

7.3.2 Secure Communication

Raw Data Strem communication can be transported via TCP and UDP. Therefore different security mechanisms is available to secure the communication. The following security protocols are currently supported:

- TLS 1.2 (see [RFC5246])
- DTLS 1.2 (see [RFC6347])
- IPSec

7.3.2.1 Creation and use of secure channels for Raw Data Streaming

[SWS_RDS_90211] Secure UDP and TCP channel creation for TLS and DTLS ⌈The Raw Data Stream software shall create secure UDP and TCP channels according to the input for all `TlsSecureComProps` as part of the `EthernetRawDataStreamMapping`.⌋*(RS_CM_00801)*

[SWS_RDS_90212] Using secure TLS, DTLS channels ⌈All communication triggered by a `RawDataStream` shall be sent via the respective secure channel according to the input. The appropriate secure channel is defined in the `TlsSecureComProps` as part of the `EthernetRawDataStreamMapping` that is mapped to an `EthernetCommunicationConnector`.⌋*(RS_CM_00801, RS_CM_00803)*

7.3.2.2 (D)TLS for Raw Data Streaming

A (D)TLS secure channel may provide authenticity, integrity and confidentiality which may be used with raw data streaming.

The TLS and DTLS implementation should support the following cipher suites:

- `TLS_PSK_WITH_NULL_SHA256` for authentic communication (see [RFC5487])
- `TLS_PSK_WITH_AES_128_GCM_SHA256` for confidential communication (see [RFC5487])

[SWS_RDS_90213] TLS secure channel for raw data streams using reliable transport ⌈A TLS secure channel shall be created and used if

- a `TlsSecureComProps` instance is part of a `EthernetRawDataStreamMapping` and is configured for transmission over “tcp” by assigning a `localTcpPort` in the `EthernetRawDataStreamMapping`

⌋*(RS_CM_00801)*

[SWS_RDS_90214] DTLS secure channel for methods using unreliable transport ⌈A DTLS secure channel shall be created and used if:

- a TlsSecureComProps instance is part of a EthernetRawDataStreamMapping and is configured for transmission over “udp” by assigning a localUdpPort in the EthernetRawDataStreamMapping

⌋(*RS_CM_00801*)

[SWS_RDS_90215] IPsec secure channel between communication nodes and Transport of Raw Data Stream communication over an IPsec security association ⌈An IPsec secure channel shall be created and used according to the requirements and constraints specified in [SWS_CM_90117] and [SWS_CM_90118], (see [6, SWS Communication Management]), by applying the EthernetRawDataStreamMapping to map to the EthernetCommunicationConnector that in turn references a NetworkEndpoint that contains an IPSecConfig.⌋(*RS_CM_00801*)

7.4 Safety

Safety protection like E2E has to be applied on application level.

7.5 Functional cluster lifecycle

7.5.1 Startup

No special startup handling is needed for the RAW Data Stream functional cluster (e.g. no state is maintained across power cycles).

7.5.2 Shutdown

No special shutdown handling is needed for the RAW Data Stream functional cluster.

8 API specification

8.1 C++ language binding

8.1.1 Raw Data Stream header file

The *Raw data stream header file* includes the data type definitions specific for the ara::rds API for Raw Data Streams.

[SWS_RDS_10488] Raw data stream header file existence ⌈The communication management shall provide a *Raw data stream header file* by using the file name `raw_data_stream.h`.⌋*(RS_CM_00001)*

[SWS_RDS_10489] Raw data stream header file namespace ⌈The C++ namespace for the data type definitions included by the *Raw data stream header file* shall be:

```
namespace ara {
namespace rds {
...
} // namespace rds
} // namespace ara
```

⌋*(RS_CM_00002)*

[SWS_RDS_10490] Data Type declarations in Raw data stream header file ⌈The *Raw data stream header file* shall include the class definitions according to [SWS_RDS_10481], [SWS_RDS_10482], [SWS_RDS_10483], [SWS_RDS_10484], [SWS_RDS_10485], [SWS_RDS_10486] and [SWS_RDS_10487].⌋*(RS_CM_00001)*

8.2 API Data Types

8.2.1 Raw Data Stream Data Types

[SWS_RDS_11300]{DRAFT} **Definition of API class ara::rds::ReadDataResult** ⌈

Kind:	struct
Header file:	#include "ara/rds/raw_data_stream.h"
Forwarding header file:	#include "ara/rds/rds_fwd.h"
Scope:	namespace ara::rds
Symbol:	ReadDataResult
Syntax:	`struct ReadDataResult {...};`
Description:	The ReadDataResult struct used as return value from ReadData().

⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_RDS_11301]{DRAFT} **Definition of API variable ara::rds::ReadDataResult::data** ⌈

Kind:	variable
Header file:	#include "ara/rds/raw_data_stream.h"
Scope:	struct ara::rds::ReadDataResult
Symbol:	data
Type:	`std::unique_ptr< ara::core::Byte[]>`
Syntax:	`std::unique_ptr<ara::core::Byte[]> data;`
Description:	std::unique pointer to the read data.

⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_RDS_11302]{DRAFT} **Definition of API variable ara::rds::ReadDataResult::numberOfBytes** ⌈

Kind:	variable
Header file:	#include "ara/rds/raw_data_stream.h"
Scope:	struct ara::rds::ReadDataResult
Symbol:	numberOfBytes
Type:	std::size_t
Syntax:	`std::size_t numberOfBytes;`
Description:	The actual number of bytes read from the stream.

⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

8.2.2 Raw Data Stream Error Types

[SWS_RDS_99025]{DRAFT} **Raw errors domain** ⌈Error domain to describe ara::core errors related to the RawDataStreamInterface `ara::rds::RawErrorDomain` shall be defined. It shall have the shortname Raw and the identifier 0x8000'0000'0000'1280.⌋*(RS_AP_00130)*

[SWS_RDS_12367] Definition of API enum ara::rds::RawErrc ⌈

Kind:	enumeration	
Header file:	#include "ara/rds/raw_error_domain.h"	
Forwarding header file:	#include "ara/rds/rds_fwd.h"	
Scope:	namespace ara::rds	
Symbol:	RawErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class RawErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kStreamNotConnected= 1	Trying to use a raw data stream without an established connection.
	kCommunication Timeout= 2	The operation was not successful and timed out.

▽

△

	kConnectionRefused= 3	The target address was not listening for connections or refused the connection request.
	kAddressNotAvailable= 4	The specified address is not available from the local machine.
	kStreamAlready Connected= 5	The specified connection is already connected.
	kConnectionClosedBy Peer= 6	Network error. The established connection has been shut down during writing (POSIX EPIPE).
	kPeerUnreachable= 7	Network error. The peer is unreachable (POSIX ENETUNREACH).
	kConnectionAborted= 8	Network error. The incoming connection was aborted (POSIX ECONNABORTED).
	kInterruptedBySignal= 9	System error. Operation interrupted by system (POSIX EINTR).
	kConnectionCreation Failed= 10	Permission to create a connection is denied. (POSIX EACCES)
	kGrantEnforcement Error= 11	Request was refused by Grant enforcement layer.
Description:	The RawErrc enumeration defines the error codes for the RawErrorDomain.	

⌋(*RS_AP_00130*, *RS_AP_00122*, *RS_AP_00127*)

[SWS_RDS_11291]{DRAFT} **Definition of API class ara::rds::RawException** ⌈

Kind:	class
Header file:	#include "ara/rds/raw_error_domain.h"
Forwarding header file:	#include "ara/rds/rds_fwd.h"
Scope:	namespace ara::rds
Symbol:	RawException
Base class:	ara::core::Exception
Syntax:	`class RawException : public ara::core::Exception {...};`
Description:	Defines a class for exceptions to be thrown by the Raw Data Streams.

⌋(*RS_AP_00130*, *RS_AP_00122*, *RS_AP_00127*)

[SWS_RDS_11292]{DRAFT} **Definition of API function ara::rds::RawException::RawException** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_error_domain.h"	
Scope:	class ara::rds::RawException	
Symbol:	RawException(ara::core::ErrorCode errorCode)	
Syntax:	`explicit RawException (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code.
Exception Safety:	noexcept	
Description:	Constructs a new RawException object containing an error code.	

⌋(*RS_AP_00120*, *RS_AP_00121*, *RS_AP_00130*, *RS_AP_00132*)

[SWS_RDS_11298]{DRAFT} **Definition of API function ara::rds::GetRawErrorDomain** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_error_domain.h"	
Scope:	namespace ara::rds	
Symbol:	GetRawErrorDomain()	
Syntax:	`constexpr ara::core::ErrorDomain & GetRawErrorDomain () noexcept;`	
Return value:	ara::core::ErrorDomain &	A reference to the global RawErrorDomain object.
Exception Safety:	noexcept	
Description:	Returns a reference to the global RawErrorDomain object.	

⌋(*RS_AP_00120*, *RS_AP_00130*, *RS_AP_00132*)

[SWS_RDS_11299]{DRAFT} **Definition of API function ara::rds::MakeErrorCode** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_error_domain.h"	
Scope:	namespace ara::rds	
Symbol:	MakeErrorCode(ara::rds::RawErrc code, ara::core::ErrorDomain::SupportDataType data)	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (ara::rds::RawErrc code, ara::core::ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	Error code number.
	data	Vendor defined data associated with the error.
Return value:	ara::core::ErrorCode	An ErrorCode object.
Exception Safety:	noexcept	
Description:	Creates an instance of ErrorCode.	

⌋(*RS_AP_00120*, *RS_AP_00121*, *RS_AP_00130*, *RS_AP_00132*)

[SWS_RDS_11293]{DRAFT} **Definition of API class ara::rds::RawErrorDomain** ⌈

Kind:	class
Header file:	#include "ara/rds/raw_error_domain.h"
Forwarding header file:	#include "ara/rds/rds_fwd.h"
Scope:	namespace ara::rds
Symbol:	RawErrorDomain
Base class:	ara::core::ErrorDomain
Syntax:	`class RawErrorDomain final : public ara::core::ErrorDomain {...};`
Unique ID:	0x8000'0000'0000'1280
Description:	Defines a class representing the Raw Data Streams error domain.

⌋(*RS_AP_00130*, *RS_AP_00122*, *RS_AP_00127*)

[SWS_RDS_11294]{DRAFT} **Definition of API function ara::rds::RawErrorDomain::RawErrorDomain** ⌈

Kind:	function
Header file:	#include "ara/rds/raw_error_domain.h"
Scope:	class ara::rds::RawErrorDomain
Symbol:	RawErrorDomain()
Syntax:	`RawErrorDomain ()=delete;`
Description:	Constructs a new RawErrorDomain object - Not allowed.

⌋*(RS_AP_00120, RS_AP_00130, RS_AP_00146)*

[SWS_RDS_11295]{DRAFT} **Definition of API function ara::rds::RawErrorDomain::Name** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_error_domain.h"	
Scope:	class ara::rds::RawErrorDomain	
Symbol:	Name()	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Raw".
Exception Safety:	noexcept	
Description:	Returns a string constant associated with RawErrorDomain.	

⌋*(RS_AP_00120, RS_AP_00130, RS_AP_00132)*

[SWS_RDS_11296]{DRAFT} **Definition of API function ara::rds::RawErrorDomain::Message** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_error_domain.h"	
Scope:	class ara::rds::RawErrorDomain	
Symbol:	Message(CodeType errorCode)	
Syntax:	`const char * Message (CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	The error code number.
Return value:	const char *	The message associated with the error code.
Exception Safety:	noexcept	
Description:	Returns the message associated with errorCode.	

⌋*(RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132)*

[SWS_RDS_11297]{DRAFT} **Definition of API function ara::rds::RawErrorDomain::ThrowAsException** ⌈

Kind:	function
Header file:	#include "ara/rds/raw_error_domain.h"
Scope:	class ara::rds::RawErrorDomain
Symbol:	ThrowAsException(const ara::core::ErrorCode &errorCode)

▽

△

Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	noexcept(false)	
Description:	Creates a new instance of RawException from errorCode and throws it as a C++ exception.	

⌋(*RS_AP_00120*, *RS_AP_00121*, *RS_AP_00130*)

8.3 RawDataStreamClient class

For the functional description of the Raw Data Stream API, see chapter 7.2.

[SWS_RDS_10481] Definition of API class ara::rds::RawDataStreamClient ⌈

Kind:	class
Header file:	#include "ara/rds/raw_data_stream.h"
Forwarding header file:	#include "ara/rds/rds_fwd.h"
Scope:	namespace ara::rds
Symbol:	RawDataStreamClient
Syntax:	`class RawDataStreamClient final {...};`
Description:	This class defines a RawDataStreamClient object for reading and writing binary data streams over a network connection.

⌋(*RS_CM_00410*, *RS_CM_00411*)

8.3.1 Special member functions

[SWS_RDS_10482] Definition of API function ara::rds::RawDataStream Client::Create ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	Create(const ara::core::InstanceSpecifier &instance)	
Syntax:	`ara::core::Result< RawDataStreamClient > Create (const ara::core::InstanceSpecifier &instance) noexcept;`	
Parameters (in):	instance	The instance specifier for the instance.
Return value:	ara::core::Result< Raw DataStreamClient >	ara::core::Result<RawDataStreamClient> The RawDataStream Client object if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k ConnectionCreation Failed	Permission to create a connection is denied. (POSIX EACCES)

▽

△

	ara::rds::RawErrc::kAddressNotAvailable	The specified address is not available from the local machine.
Description:	Named exception-less constructor that takes an instance Specifier qualifying the wanted network binding and parameters for the instance. If Remote Unicast Credentials (TCP or UDP) are defined for the client, the constructor shall create an endpoint for the communication, and store the handle in the created RawDataStream Client object, to be used in the Read- and Write-operations for the RawDataStreamClient (for 1:1 use cases). If Multicast Credentials (UDP) are defined for the client, the constructor shall create an endpoint for the communication, bind and join the multicast address and port specified in the Multicast Credentials. For 1:N use cases this endpoint shall be used when RawDataStreamsClient.ReadData() is called, otherwise (for 1:1 use cases), the unicast endpoint shall be used for reading data.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*)

[SWS_RDS_10483] Definition of API function ara::rds::RawDataStream Client::~RawDataStreamClient ⌈

Kind:	function
Header file:	#include "ara/rds/raw_data_stream.h"
Scope:	class ara::rds::RawDataStreamClient
Symbol:	~RawDataStreamClient()
Syntax:	`~RawDataStreamClient () noexcept;`
Exception Safety:	noexcept
Description:	Destructor of the RawDataStreamClient that deletes the RawDataStreamClient instance. If the connection is still open, the connection is closed and shut down (calling Shutdown()) before destroying the RawDataStreamClient object.

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*)

[SWS_RDS_11303]{DRAFT} **Definition of API function ara::rds::RawDataStream Client::RawDataStreamClient** ⌈

Kind:	function
Header file:	#include "ara/rds/raw_data_stream.h"
Scope:	class ara::rds::RawDataStreamClient
Symbol:	RawDataStreamClient(const RawDataStreamClient &)
Syntax:	`RawDataStreamClient (const RawDataStreamClient &)=delete;`
Description:	Copy constructor of the RawDataStreamClient - not allowed.

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*, *RS_AP_00147*)

[SWS_RDS_11304]{DRAFT} **Definition of API function ara::rds::RawDataStream Client::operator=** ⌈

Kind:	function
Header file:	#include "ara/rds/raw_data_stream.h"
Scope:	class ara::rds::RawDataStreamClient
Symbol:	operator=(const RawDataStreamClient &)

▽

△

Syntax:	`RawDataStreamClient & operator= (const RawDataStreamClient &)=delete;`
Description:	Copy assignment operator of the RawDataStreamClient - not allowed.

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*, *RS_AP_00147*)

[SWS_RDS_11305]{DRAFT} **Definition of API function ara::rds::RawDataStream Client::RawDataStreamClient** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	RawDataStreamClient(RawDataStreamClient &&other)	
Syntax:	`RawDataStreamClient (RawDataStreamClient &&other) noexcept;`	
Parameters (in):	other	The RawDataStreamClient object to be moved.
Exception Safety:	noexcept	
Description:	Move constructor of the RawDataStreamClient.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*, *RS_AP_00147*)

[SWS_RDS_11306]{DRAFT} **Definition of API function ara::rds::RawDataStream Client::operator=** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	operator=(RawDataStreamClient &&other)	
Syntax:	`RawDataStreamClient & operator= (RawDataStreamClient &&other) & noexcept;`	
Parameters (in):	other	The RawDataStreamClient object to be moved.
Return value:	RawDataStreamClient &	–
Exception Safety:	noexcept	
Description:	Move assignment operator of the RawDataStreamClient.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*, *RS_AP_00147*)

8.3.2 Connect function

[SWS_RDS_10484] Definition of API function ara::rds::RawDataStream Client::Connect ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	Connect()	
Syntax:	`ara::core::Result< void > Connect () noexcept;`	
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.

▽

△

Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k ConnectionRefused	The connection was refused by target.
	ara::rds::RawErrc::k AddressNotAvailable	The specified address is not available from the local machine.
	ara::rds::RawErrc::k StreamAlready Connected	The specified connection is already connected.
	ara::rds::RawErrc::kPeer Unreachable	The peer is unreachable by the network.
	ara::rds::RawErrc::k InterruptedBySignal	The operation was interrupted by the system.
	ara::rds::RawErrc::k GrantEnforcementError	Request was refused by Grant enforcement layer.
Description:	Sets up a unicast socket connection for the RawDataStream defined by the instance, and establishes a connection to the TCP server. In the case of UDP, no connection is established. Incoming and outgoing packets are restricted to the specified address. The socket endpoints and attributes are specified in the manifest which is accessed through the InstanceSpecifer provided in the constructor. If TLS security protocol is configured for the socket connection, the TLS/DTLS connection shall be initialized here.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

Note: For TLS/DTLS connection with Raw Data Streaming see also chapter 7.3.2.2.

[SWS_RDS_11307]{DRAFT} **Definition of API function ara::rds::RawDataStream Client::Connect** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	Connect(std::chrono::milliseconds timeout)	
Syntax:	`ara::core::Result< void > Connect (std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	timeout	Timeout value for this operation.
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k ConnectionRefused	The connection was refused by target.
	ara::rds::RawErrc::k AddressNotAvailable	The specified address is not available from the local machine.
	ara::rds::RawErrc::k CommunicationTimeout	The connect operation timed out.
	ara::rds::RawErrc::k StreamAlready Connected	The specified connection is already connected.
	ara::rds::RawErrc::kPeer Unreachable	The peer is unreachable by the network.
	ara::rds::RawErrc::k InterruptedBySignal	The operation was interrupted by the system.

▽

△

Description:	Sets up a unicast socket connection for the RawDataStream defined by the instance, and establishes a connection to the TCP server. In the case of UDP, no connection is established. Incoming and outgoing packets are restricted to the specified address. The socket endpoints and attributes are specified in the manifest which is accessed through the InstanceSpecifer provided in the constructor. If TLS security protocol is configured for the socket connection, the TLS/DTLS connection shall be initialized here.

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

8.3.3 Shutdown function

[SWS_RDS_10485] Definition of API function ara::rds::RawDataStream Client::Shutdown ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	Shutdown()	
Syntax:	`ara::core::Result< void > Shutdown () noexcept;`	
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k StreamNotConnected	Trying to shutdown a RawDataStream without an established connection.
	ara::rds::RawErrc::k InterruptedBySignal	The operation was interrupted by the system.
Description:	Closes the socket connection for the RawDataStream defined by the instance. Both the receiving and the sending part of the socket connection shall be shut down. For TCP, the full-duplex connection shall be shut down disallowing further receptions and transmissions, before closing the socket.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

8.3.4 ReadData function

[SWS_RDS_10486] Definition of API function ara::rds::RawDataStream Client::ReadData ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	ReadData(std::size_t maxLength)	
Syntax:	`ara::core::Result< ReadDataResult > ReadData (std::size_t maxLength) noexcept;`	
Parameters (in):	maxLength	The requested maximum number of bytes to read from the stream.
Return value:	ara::core::Result< Read DataResult >	a struct of type ReadDataResult if succesful, otherwise an error code indicating the error.

▽

△

Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k StreamNotConnected	Trying to read from a stream without an established connection.
	ara::rds::RawErrc::k InterruptedBySignal	The operation was interrupted by the system.
Description:	Requests to read a number of bytes of data from the socket connection for the RawDataStream defined by the instance. If Multicast Credentials are defined for the client, the data shall be read from the multicast socket created in the constructor (for 1:N use cases), otherwise the data shall be read from the unicast TCP socket connection set up in Connect() (for 1:1 TCP unicast use case), or the unicast UDP socket created in the constructor (for 1:1 UDP unicast use case). For efficiency, the zero-copy semantics of std::unique_ptr is used, which means that the ownership of the allocated memory of the read data is transferred to the application in the Read DataResult.data value.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_11309]{DRAFT} **Definition of API function ara::rds::RawDataStream Client::ReadData** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	ReadData(std::size_t maxLength, std::chrono::milliseconds timeout)	
Syntax:	`ara::core::Result< ReadDataResult > ReadData (std::size_t maxLength, std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	maxLength	The requested maximum number of bytes to read from the stream.
	timeout	Timeout value for this operation.
Return value:	ara::core::Result< ReadDataResult >	a struct of type ReadDataResult if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k StreamNotConnected	Trying to read from a stream without an established connection.
	ara::rds::RawErrc::k CommunicationTimeout	No data was read until the timeout expired.
	ara::rds::RawErrc::k InterruptedBySignal	The operation was interrupted by the system.
Description:	Requests to read a number of bytes of data from the socket connection for the RawDataStream defined by the instance. If Multicast Credentials are defined for the client, the data shall be read from the multicast socket created in the constructor (for 1:N use cases), otherwise the data shall be read from the unicast TCP socket connection set up in Connect() (for 1:1 TCP unicast use case), or the unicast UDP socket created in the constructor (for 1:1 UDP unicast use case). For efficiency, the zero-copy semantics of std::unique_ptr is used, which means that the ownership of the allocated memory of the read data is transferred to the application in the Read DataResult.data value.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

8.3.5 WriteData function

[SWS_RDS_10487] Definition of API function ara::rds::RawDataStream Client::WriteData ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	WriteData(std::unique_ptr< ara::core::Byte[]> data, std::size_t maxLength)	
Syntax:	`ara::core::Result< std::size_t > WriteData (std::unique_ptr< ara::core::Byte[]> data, std::size_t maxLength) noexcept;`	
Parameters (in):	data	std::unique pointer to the byte array to send.
	maxLength	The requested maximum number of bytes to write to the stream.
Return value:	ara::core::Result< std::size_t >	the actual number of bytes written if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k StreamNotConnected	Trying to write to a stream without an established connection.
	ara::rds::RawErrc::k ConnectionClosedBy Peer	The established connection has been shut down during writing.
	ara::rds::RawErrc::k InterruptedBySignal	The operation was interrupted by the system.
Description:	Requests to write of a number of bytes to the the socket connection for the RawDataStream defined by the instance (for 1:1 use cases). If Multicast Credentials are defined for the client reading of data, a single socket can be used for both multicast reading and unicast writing (for use case 1:N UDP + 1:1 UDP, Server sends data via multicast, and a client sends control data via unicast). For efficiency, the zero-copy semantics of std::unique_ptr is used.	

⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_RDS_11310]{DRAFT} **Definition of API function ara::rds::RawDataStream Client::WriteData** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamClient	
Symbol:	WriteData(std::unique_ptr< ara::core::Byte[]> data, std::size_t maxLength, std::chrono::milliseconds timeout)	
Syntax:	`ara::core::Result< std::size_t > WriteData (std::unique_ptr< ara::core::Byte[]> data, std::size_t maxLength, std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	data	std::unique pointer to the byte array to send.
	maxLength	The requested maximum number of bytes to write to the stream.
	timeout	Timeout value for this operation.
Return value:	ara::core::Result< std::size_t >	the actual number of bytes written if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k StreamNotConnected	Trying to write to a stream without an established connection.

▽

△

	ara::rds::RawErrc::kCommunicationTimeout	No data was written until the timeout expired.
	ara::rds::RawErrc::kConnectionClosedByPeer	The established connection has been shut down during writing.
	ara::rds::RawErrc::kInterruptedBySignal	The operation was interrupted by the system.
Description:	Requests to write a number of bytes to the the socket connection for the RawDataStream defined by the instance. If Multicast Credentials are defined for the client reading of data, a single socket can be used for both multicast reading and unicast writing (for use case 1:N UDP + 1:1 UDP, Server sends data via multicast, and a client sends control data via unicast). For efficiency, the zero-copy semantics of std::unique_ptr is used.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

8.4 RawDataStreamServer class

For the functional description of the Raw Data Stream API, see chapter 7.2.

[SWS_RDS_11311]{DRAFT} **Definition of API class ara::rds::RawDataStreamServer** ⌈

Kind:	class
Header file:	#include "ara/rds/raw_data_stream.h"
Forwarding header file:	#include "ara/rds/rds_fwd.h"
Scope:	namespace ara::rds
Symbol:	RawDataStreamServer
Syntax:	`class RawDataStreamServer final {...};`
Description:	This class defines a RawDataStreamServer object for reading and writing binary data streams over a network connection.

⌋(*RS_CM_00410*, *RS_CM_00411*)

8.4.1 Special member functions

[SWS_RDS_11312]{DRAFT} **Definition of API function ara::rds::RawDataStreamServer::Create** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	Create(const ara::core::InstanceSpecifier &instance)	
Syntax:	`ara::core::Result< RawDataStreamServer > Create (const ara::core::InstanceSpecifier &instance) noexcept;`	
Parameters (in):	instance	The instance specifier for the instance.

▽

△

Return value:	ara::core::Result< Raw DataStreamServer >	ara::core::Result<RawDataStreamServer> The RawDataStream Server object if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k ConnectionCreation Failed	Permission to create a connection is denied. (POSIX EACCES)
	ara::rds::RawErrc::k AddressNotAvailable	The specified address is not available from the local machine.
	ara::rds::RawErrc::k StreamAlready Connected	The specified connection is already connected.
Description:	Named exception-less constructor that takes an instance Specifier qualifying the wanted network credentials (UDP or TCP) for the instance. A socket shall be created and bound to the address and port specified in the local credentials. In case of TCP it shall also mark the socket as passive and listen for connections (for use case 1:1 TCP unicast). If Remote Unicast Credentials (UDP) are defined for the server, the constructor shall create an endpoint for the communication, and store the handle in the created RawDataStreamServer object, to be used in the Read and Write- operations for the RawDataStreamServer (for use case 1:1 UDP unicast). If Multicast Credentials (UDP) are defined for the server, the constructor shall create an endpoint for the remote communication, bind and join the multicast address and port specified in the MulticastCredentials. In this case, this endpoint shall be used when RawDataStreams Server.WriteData() is called (for 1:N use cases), otherwise the unicast endpoint shall be used for writing data (for 1:1 use cases).	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*)

[SWS_RDS_11313]{DRAFT} Definition of API function ara::rds::RawDataStream Server::~RawDataStreamServer ⌈

Kind:	function
Header file:	#include "ara/rds/raw_data_stream.h"
Scope:	class ara::rds::RawDataStreamServer
Symbol:	~RawDataStreamServer()
Syntax:	`~RawDataStreamServer () noexcept;`
Exception Safety:	noexcept
Description:	Destructor of the RawDataStreamServer that deletes the RawDataStreamServer instance. If the connection is still open, the connection is closed and shut down (calling Shutdown()) before destroying the RawDataStreamClient object.

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*)

[SWS_RDS_11314]{DRAFT} Definition of API function ara::rds::RawDataStream Server::RawDataStreamServer ⌈

Kind:	function
Header file:	#include "ara/rds/raw_data_stream.h"
Scope:	class ara::rds::RawDataStreamServer
Symbol:	RawDataStreamServer(const RawDataStreamServer &)
Syntax:	`RawDataStreamServer (const RawDataStreamServer &)=delete;`
Description:	Copy constructor of the RawDataStreamServer - not allowed.

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*)

[SWS_RDS_11315]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::operator=** ⌈

Kind:	function
Header file:	#include "ara/rds/raw_data_stream.h"
Scope:	class ara::rds::RawDataStreamServer
Symbol:	operator=(const RawDataStreamServer &)
Syntax:	`RawDataStreamServer & operator= (const RawDataStreamServer &)=delete;`
Description:	Copy assignment operator of the RawDataStreamServer - not allowed.

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*)

[SWS_RDS_11316]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::RawDataStreamServer** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	RawDataStreamServer(RawDataStreamServer &&other)	
Syntax:	`RawDataStreamServer (RawDataStreamServer &&other) noexcept;`	
Parameters (in):	other	The RawDataStreamServer object to be moved.
Exception Safety:	noexcept	
Description:	Move constructor of the RawDataStreamServer.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*, *RS_AP_00147*)

[SWS_RDS_11317]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::operator=** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	operator=(RawDataStreamServer &&other)	
Syntax:	`RawDataStreamServer & operator= (RawDataStreamServer &&other) & noexcept;`	
Parameters (in):	other	The RawDataStreamServer object to be moved.
Return value:	RawDataStreamServer &	–
Exception Safety:	noexcept	
Description:	Move assignment operator of the RawDataStreamServer.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*, *RS_AP_00145*, *RS_AP_00147*)

8.4.2 WaitForConnection function

[SWS_RDS_11318]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::WaitForConnection** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	WaitForConnection()	
Syntax:	`ara::core::Result< void > WaitForConnection () noexcept;`	
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::kConnectionAborted	The incoming connection was aborted by the network.
	ara::rds::RawErrc::kInterruptedBySignal	The operation was interrupted by the system.
	ara::rds::RawErrc::kGrantEnforcementError	Request was refused by Grant enforcement layer.
Description:	Enables the RawDataStreamServer instance for incoming connections. For TCP the constructor marks the socket as ready to accept connection requests from a client (see SWS_RDS_11312), and WaitForConnection() waits to accept an incoming connection request. In the case of UDP, no connection is established, and the operation shall return with no action.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_11319]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::WaitForConnection** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	WaitForConnection(std::chrono::milliseconds timeout)	
Syntax:	`ara::core::Result< void > WaitForConnection (std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	timeout	Timeout value for this operation.
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::kCommunicationTimeout	The WaitForConnection operation timed out.
	ara::rds::RawErrc::kConnectionAborted	The incoming connection was aborted by the network.
	ara::rds::RawErrc::kInterruptedBySignal	The operation was interrupted by the system
Description:	Enables the RawDataStreamServer instance for incoming connections. For TCP the constructor marks the socket as ready to accept connection requests from a client (see SWS_RDS_11312), and WaitForConnection() waits to accept an incoming connection request. In the case of UDP, no connection is established, and the operation shall return with no action.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

8.4.3 Shutdown function

[SWS_RDS_11320]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::Shutdown** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	Shutdown()	
Syntax:	`ara::core::Result< void > Shutdown () noexcept;`	
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::kStreamNotConnected	Trying to shutdown a RawDataStream without an established connection.
	ara::rds::RawErrc::kInterruptedBySignal	The operation was interrupted by the system.
Description:	Closes the socket connection for the RawDataStream defined by the instance. Both the receiving and the sending part of the socket connection shall be shut down. For TCP, the full-duplex connection shall be shut down disallowing further receptions and transmissions, before closing the socket.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

8.4.4 ReadData function

[SWS_RDS_11322]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::ReadData** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	ReadData(std::size_t maxLength)	
Syntax:	`ara::core::Result< ReadDataResult > ReadData (std::size_t maxLength) noexcept;`	
Parameters (in):	maxLength	The requested maximum number of bytes to read from the stream.
Return value:	ara::core::Result< ReadDataResult >	a struct of type ReadDataResult if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::kStreamNotConnected	Trying to read from a stream without an established connection.
	ara::rds::RawErrc::kInterruptedBySignal	The operation was interrupted by the system.
Description:	Requests to read a number of bytes of data from the Unicast socket connection for the RawData Stream defined by the instance. For efficiency, the zero-copy semantics of std::unique_ptr is used, which means that the ownership of the allocated memory of the read data is transferred to the application in the ReadDataResult.data value.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_11323]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::ReadData** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	ReadData(std::size_t maxLength, std::chrono::milliseconds timeout)	
Syntax:	`ara::core::Result< ReadDataResult > ReadData (std::size_t maxLength, std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	maxLength	The requested maximum number of bytes to read from the stream.
	timeout	Parameter to assign a timeout for this operation.
Return value:	ara::core::Result< ReadDataResult >	a struct of type ReadDataResult.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::kStreamNotConnected	Trying to read from a stream without an established connection.
	ara::rds::RawErrc::kCommunicationTimeout	No data was read until the timeout expired.
	ara::rds::RawErrc::kInterruptedBySignal	The operation was interrupted by the system.
Description:	Requests to read a number of bytes of data from the unicast socket connection for the RawData Stream defined by the instance. For efficiency, the zero-copy semantics of std::unique_ptr is used, which means that the ownership of the allocated memory of the read data is transferred to the application in the ReadDataResult.data value.	

⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

8.4.5 WriteData function

[SWS_RDS_11324]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::WriteData** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	WriteData(std::unique_ptr< ara::core::Byte[]> data, std::size_t maxLength)	
Syntax:	`ara::core::Result< std::size_t > WriteData (std::unique_ptr< ara::core::Byte[]> data, std::size_t maxLength) noexcept;`	
Parameters (in):	data	std::unique pointer to the byte array to send.
	maxLength	The requested maximum number of bytes to write to the stream.
Return value:	ara::core::Result< std::size_t >	the actual number of bytes written if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::kStreamNotConnected	Trying to write to a stream without an established connection.
	ara::rds::RawErrc::kConnectionClosedByPeer	The established connection has been shut down during writing.

▽

△

	ara::rds::RawErrc::k InterruptedBySignal	The operation was interrupted by the system.
Description:	Requests to write a number of bytes to the the socket connection for the RawDataStream defined by the instance. If Remote Multicast Credentials are defined for the server, the data shall be written to the multicast socket created in the constructor (for 1:N use cases). Otherwise in case of TCP, the data shall be written to the unicast socket connection set up in WaitForConnection() (for 1:1 TCP unicast use case). In case of UDP the data shall be written to the unicast socket created in the constructor (1:1 UDP unicast). For efficiency, the zero-copy semantics of std::unique_ptr is used.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_RDS_11325]{DRAFT} **Definition of API function ara::rds::RawDataStream Server::WriteData** ⌈

Kind:	function	
Header file:	#include "ara/rds/raw_data_stream.h"	
Scope:	class ara::rds::RawDataStreamServer	
Symbol:	WriteData(std::unique_ptr< ara::core::Byte[]> data, std::size_t maxLength, std::chrono::milliseconds timeout)	
Syntax:	`ara::core::Result< std::size_t > WriteData (std::unique_ptr< ara::core::Byte[]> data, std::size_t maxLength, std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	data	std::unique pointer to the byte array to send.
	maxLength	The requested maximum number of bytes to write to the stream.
	timeout	Parameter to assign a timeout for this operation.
Return value:	ara::core::Result< std::size_t >	the actual number of bytes written if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::rds::RawErrc::k StreamNotConnected	Trying to write to a stream without an established connection.
	ara::rds::RawErrc::k CommunicationTimeout	No data was written until the timeout expired.
	ara::rds::RawErrc::k ConnectionClosedBy Peer	The established connection has been shut down during writing.
	ara::rds::RawErrc::k InterruptedBySignal	The operation was interrupted by the system.
Description:	Requests to write a number of bytes to the the socket connection for the RawDataStream defined by the instance. If Remote Multicast Credentials are defined for the server, the data shall be written to the multicast socket created in the constructor (for 1:N use cases). Otherwise in case of TCP, the data shall be written to the unicast socket connection set up in WaitForConnection() (for 1:1 TCP unicast use case). In case of UDP the data shall be written to the unicast socket created in the constructor (1:1 UDP unicast). For efficiency, the zero-copy semantics of std::unique_ptr is used.	

⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

9 Service Interfaces

`RAW Data Stream` does not contain any Service Interfaces.

A Mentioned Manifest Elements

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Chapter is generated.

Class	***AbstractRawDataStreamEthernetCredentials*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class serves as an abstract base class for the configuration of network credentials.			
Base	*ARObject*, *Describable*			
Subclasses	RawDataStreamEthernetTcpUdpCredentials, RawDataStreamEthernetUdpCredentials			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
ipV4Address	Ip4AddressString	0..1	attr	This attribute describes the IP V4 address of the remote server.
ipV6Address	Ip6AddressString	0..1	attr	This attribute describes the IP V6 address of the remote server.
udpPort	PositiveInteger	0..1	attr	This attribute represents the configuration of a UDP port number.

Table A.1: AbstractRawDataStreamEthernetCredentials

Class	**EthernetCommunicationConnector**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Ethernet specific attributes to the CommunicationConnector.			
Base	*ARObject*, *CommunicationConnector*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	EcuInstance.connector, MachineDesign.communicationConnector			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
apApplication Endpoint	ApApplicationEndpoint	*	aggr	Collection of Application Addresses that are used on the CommunicationConnector.
canXlProps	CanXlProps	*	ref	If the Ethernet frames handled by this Ethernet CommunicationConnector are tunneled through CAN XL, then this reference shall refer the CanXlProps which contains the specific configuration parameters of the CAN XL controller of the physical CAN XL connection to be used for tunneling.
maximum Transmission Unit	PositiveInteger	0..1	attr	This attribute specifies the maximum transmission unit in bytes.
neighborCache Size	PositiveInteger	0..1	attr	This attribute specifies the size of neighbor cache or ARP table in units of entries.
pathMtu Enabled	Boolean	0..1	attr	If enabled the IPv4/IPv6 processes incoming ICMP "Packet Too Big" messages and stores a MTU value for each destination address.
pathMtuTimeout	TimeValue	0..1	attr	If this value is >0 the IPv4/IPv6 will reset the MTU value stored for each destination after n seconds.
unicastNetwork Endpoint	NetworkEndpoint	*	ref	Network Endpoint that defines the IPAddress of the machine.

Table A.2: EthernetCommunicationConnector

Class	**EthernetRawDataStreamClientMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class represents the ability to map a client PortPrototype to a Ethernet-based communication channel. **Tags:** atp.recommendedPackage=RawDataStreamingMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *EthernetRawDataStreamMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *RawDataStreamMapping*, *Referrable*, *Uploadable DeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
remoteServer Config	EthernetRawData StreamRemoteServer Config	0..1	aggr	This aggregation is used to configure the credentials of the remote server.

Table A.3: EthernetRawDataStreamClientMapping

Class	**EthernetRawDataStreamGrant**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to define the IAM configuration for a EthernetRawDataStream on deployment level. **Tags:** atp.Status=candidate atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *RawDataStreamGrant*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackage Element*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
ethernetRaw DataStream Mapping	EthernetRawData StreamMapping	0..1	ref	This reference identifies the applicable EthernetRawData Stream to which the enclosing EthernetRawDataStream Grant shall apply. **Tags:** atp.Status=candidate

Table A.4: EthernetRawDataStreamGrant

Class	**EthernetRawDataStreamLocalEndpointConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class has the ability to act as a wrapper for the configuration of the remote endpoint in the context of a raw data stream mapping.			
Base	*ARObject*			
Aggregated by	*EthernetRawDataStreamMapping*.localEndpointConfig			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
localComm Connector	EthernetCommunication Connector	0..1	ref	This attribute represents the CommunicationConnector taken for socket-based data communication.
localTcpPort	ApApplicationEndpoint	0..1	ref	This aggregation represents the configuration of a local TCP port number.
localUdpPort	ApApplicationEndpoint	0..1	ref	This aggregation represents the configuration of a local unicast UDP port number.

Table A.5: EthernetRawDataStreamLocalEndpointConfig

Class	***EthernetRawDataStreamMapping*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class serves as the abstract bases class for the ability to map a PortPrototype to a Ethernet-based communication channel.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *RawDataStreamMapping*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Subclasses	EthernetRawDataStreamClientMapping, EthernetRawDataStreamServerMapping			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
localEndpoint Config	EthernetRawData StreamLocalEndpoint Config	0..1	aggr	This aggregation is used to configure the credentials of the endpoint.
socketOption	String	*	attr	This attribute represents the ability to specify non-formal socket options that might only be valid for specific platforms. AUTOSAR does not define a standardized meaning for the possible values of this attribute.
tlsSecureCom Props	TlsSecureComProps	0..1	ref	This reference provides the ability to define TLS-related properties for the enclosing SocketRawDataStream Mapping.

Table A.6: EthernetRawDataStreamMapping

Class	**EthernetRawDataStreamRemoteClientConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class has the ability to act as a wrapper for the configuration of the remote server in the context of a raw data stream client mapping.			
Base	*ARObject*			
Aggregated by	EthernetRawDataStreamServerMapping.remoteClientConfig			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
multicast Credentials	RawDataStream EthernetUdpCredentials	0..1	aggr	This aggregation represents the configuration of multicast credentials for communication with a remote raw data stream client.
unicastUdp Credentials	RawDataStream EthernetUdpCredentials	0..1	aggr	This aggregation represents the configuration of a remote raw data stream client that communicates via unicast over UDP.

Table A.7: EthernetRawDataStreamRemoteClientConfig

Class	**EthernetRawDataStreamRemoteServerConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class has the ability to act as a wrapper for the configuration of the remote server in the context of a raw data stream client mapping.			
Base	*ARObject*			
Aggregated by	EthernetRawDataStreamClientMapping.remoteServerConfig			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
multicast Credentials	RawDataStream EthernetUdpCredentials	0..1	aggr	This aggregation represents the configuration of multicast credentials for communication with a remote raw data stream server.
unicast Credentials	RawDataStream EthernetTcpUdp Credentials	0..1	aggr	This meta-class represents the ability to map a server PortPrototype to a Ethernet-based communication channel.

Table A.8: EthernetRawDataStreamRemoteServerConfig

Class	**EthernetRawDataStreamServerMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class represents the ability to map a server PortPrototype to a Ethernet-based communication channel. **Tags:** atp.recommendedPackage=RawDataStreamingMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *EthernetRawDataStreamMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *RawDataStreamMapping*, *Referrable*, *Uploadable DeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
remoteClient Config	EthernetRawData StreamRemoteClient Config	0..1	aggr	This aggregation is used to configure the credentials of the remote client.

Table A.9: EthernetRawDataStreamServerMapping

Class	**IPSecConfig**			
Package	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
Note	IPsec is a protocol that is designed to provide "end-to-end" cryptographically-based security for IP network connections.			
Base	*ARObject*			
Aggregated by	NetworkEndpoint.ipSecConfig			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
ipSecConfig Props	IPSecConfigProps	0..1	ref	Global IPsec configuration settings that are valid for all IPSecRules that are defined on the NetworkEndpoint.
ipSecRule	IPSecRule	*	aggr	IPSec rules and filters that are defined in the IPSecConfig for a specific NetworkEndpoint.

Table A.10: IPSecConfig

Class	**NetworkEndpoint**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	The network endpoint defines the network addressing (e.g. IP-Address or MAC multicast address).			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	EthernetPhysicalChannel.networkEndpoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
fullyQualified DomainName	String	0..1	attr	Defines the fully qualified domain name (FQDN) e.g. some.example.host.
ipSecConfig	IPSecConfig	0..1	aggr	Optional IPSec configuration that provides security services for IP packets.
network Endpoint Address	NetworkEndpoint Address	*	aggr	Definition of a Network Address. **Tags:** xml.name Plural=NETWORK-ENDPOINT-ADDRESSES
priority	PositiveInteger	0..1	attr	Defines the frame priority where values from 0 (best effort) to 7 (highest) are allowed.

Table A.11: NetworkEndpoint

Class	**Process**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class provides information required to execute the referenced `Executable`. **Tags:** atp.recommendedPackage=Processes			
Base	*ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
design	ProcessDesign	0..1	ref	This reference represents the identification of the design-time representation for the Process that owns the reference.
executable	Executable	*	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef
functionCluster Affiliation	String	0..1	attr	This attribute specifies which functional cluster the Process is affiliated with.
numberOf RestartAttempts	PositiveInteger	0..1	attr	This attribute defines how often a process shall be restarted if the start fails. numberOfRestartAttempts = "0" OR Attribute not existing, start once numberOfRestartAttempts = "1", start a second time
preMapping	Boolean	0..1	attr	This attribute describes whether the executable is preloaded into the memory.
processState Machine	ModeDeclarationGroup Prototype	0..1	aggr	Set of Process States that are defined for the process.
securityEvent	SecurityEventDefinition	*	ref	The reference identifies the collection of SecurityEvents that can be reported by the Process. **Stereotypes:** atpSplitable; atpUriDef **Tags:** atp.Splitkey=securityEvent atp.Status=candidate
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations.

Table A.12: Process

Class	**RawDataStreamEthernetTcpUdpCredentials**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This-meta-class represents the ability to create a configuration of network credentials for a raw data stream connection over TCP and UDP (inherited from base class).			
Base	*ARObject*, *AbstractRawDataStreamEthernetCredentials*, *Describable*			
Aggregated by	EthernetRawDataStreamRemoteServerConfig.unicastCredentials			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
tcpPort	PositiveInteger	0..1	attr	This attribute represents the configuration of a TCP port number.

Table A.13: RawDataStreamEthernetTcpUdpCredentials

Class	**RawDataStreamEthernetUdpCredentials**
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping
Note	This-meta-class represents the ability to create a configuration of network credentials for a raw data stream connection over UDP.

▽

△

Class	RawDataStreamEthernetUdpCredentials			
Base	*ARObject*, *AbstractRawDataStreamEthernetCredentials*, *Describable*			
Aggregated by	EthernetRawDataStreamRemoteClientConfig.multicastCredentials, EthernetRawDataStreamRemote ClientConfig.unicastUdpCredentials, EthernetRawDataStreamRemoteServerConfig.multicastCredentials			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.14: RawDataStreamEthernetUdpCredentials

Class	***RawDataStreamMapping*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class acts as an abstract base class for mapping raw data streams to the application software.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Subclasses	*EthernetRawDataStreamMapping*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
deployment	RawDataStream Deployment	0..1	ref	This reference identifies the applicable RawDataStream Deployment.
portPrototype	RPortPrototype	0..1	iref	Reference to a specific PortPrototype that represents the raw data stream to the application. **Stereotypes:** atpUriDef **InstanceRef implemented by:** RPortPrototypeIn ExecutableInstanceRef
process	Process	0..1	ref	Reference to the Process in which the Executable that contains the SoftwareComponent and the referenced Port Prototype is executed.

Table A.15: RawDataStreamMapping

Class	TlsSecureComProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	Configuration of the Transport Layer Security protocol (TLS). **Tags:** atp.recommendedPackage=SecureComProps			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *SecureComProps*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
keyExchange	CryptoServicePrimitive	*	ref	This reference identifies the shared (i.e. applicable for each of the aggregated cipher suites) crypto service primitive for the execution of key exchange during the handshake phase.
tlsCipherSuite	TlsCryptoCipherSuite	*	aggr	Collection of supported cipher suites that are used to negotiate the security settings for a network connection defined by the ServiceInstanceToMachineMapping.

Table A.16: TlsSecureComProps

B Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

B.1 Traceable item history of this document according to AUTOSAR Release R23-11

B.1.1 Added Specification Items in R23-11

Number	Heading
[SWS_RDS_10476]	Defining a RawDataStream
[SWS_RDS_10477]	Connect stream link
[SWS_RDS_10478]	Shutdown stream link
[SWS_RDS_10479]	Read data from stream
[SWS_RDS_10480]	Write data to stream
[SWS_RDS_10481]	Definition of API class ara::rds::RawDataStreamClient
[SWS_RDS_10482]	Definition of API function ara::rds::RawDataStreamClient::Create
[SWS_RDS_10483]	Definition of API function ara::rds::RawDataStreamClient::~RawDataStream Client
[SWS_RDS_10484]	Definition of API function ara::rds::RawDataStreamClient::Connect
[SWS_RDS_10485]	Definition of API function ara::rds::RawDataStreamClient::Shutdown
[SWS_RDS_10486]	Definition of API function ara::rds::RawDataStreamClient::ReadData
[SWS_RDS_10487]	Definition of API function ara::rds::RawDataStreamClient::WriteData
[SWS_RDS_10488]	Raw data stream header file existence
[SWS_RDS_10489]	Raw data stream header file namespace
[SWS_RDS_10490]	Data Type declarations in Raw data stream header file
[SWS_RDS_10508]	Destructor behavior when Shutdown stream link
[SWS_RDS_11291]	Definition of API class ara::rds::RawException
[SWS_RDS_11292]	Definition of API function ara::rds::RawException::RawException
[SWS_RDS_11293]	Definition of API class ara::rds::RawErrorDomain
[SWS_RDS_11294]	Definition of API function ara::rds::RawErrorDomain::RawErrorDomain
[SWS_RDS_11295]	Definition of API function ara::rds::RawErrorDomain::Name
[SWS_RDS_11296]	Definition of API function ara::rds::RawErrorDomain::Message
[SWS_RDS_11297]	Definition of API function ara::rds::RawErrorDomain::ThrowAsException
[SWS_RDS_11298]	Definition of API function ara::rds::GetRawErrorDomain
[SWS_RDS_11299]	Definition of API function ara::rds::MakeErrorCode
[SWS_RDS_11300]	Definition of API class ara::rds::ReadDataResult

△

Number	Heading
[SWS_RDS_11301]	Definition of API variable ara::rds::ReadDataResult::data
[SWS_RDS_11302]	Definition of API variable ara::rds::ReadDataResult::numberOfBytes
[SWS_RDS_11303]	Definition of API function ara::rds::RawDataStreamClient::RawDataStream Client
[SWS_RDS_11304]	Definition of API function ara::rds::RawDataStreamClient::operator=
[SWS_RDS_11305]	Definition of API function ara::rds::RawDataStreamClient::RawDataStream Client
[SWS_RDS_11306]	Definition of API function ara::rds::RawDataStreamClient::operator=
[SWS_RDS_11307]	Definition of API function ara::rds::RawDataStreamClient::Connect
[SWS_RDS_11309]	Definition of API function ara::rds::RawDataStreamClient::ReadData
[SWS_RDS_11310]	Definition of API function ara::rds::RawDataStreamClient::WriteData
[SWS_RDS_11311]	Definition of API class ara::rds::RawDataStreamServer
[SWS_RDS_11312]	Definition of API function ara::rds::RawDataStreamServer::Create
[SWS_RDS_11313]	Definition of API function ara::rds::RawDataStreamServer::~RawData StreamServer
[SWS_RDS_11314]	Definition of API function ara::rds::RawDataStreamServer::RawDataStream Server
[SWS_RDS_11315]	Definition of API function ara::rds::RawDataStreamServer::operator=
[SWS_RDS_11316]	Definition of API function ara::rds::RawDataStreamServer::RawDataStream Server
[SWS_RDS_11317]	Definition of API function ara::rds::RawDataStreamServer::operator=
[SWS_RDS_11318]	Definition of API function ara::rds::RawDataStreamServer::WaitFor Connection
[SWS_RDS_11319]	Definition of API function ara::rds::RawDataStreamServer::WaitFor Connection
[SWS_RDS_11320]	Definition of API function ara::rds::RawDataStreamServer::Shutdown
[SWS_RDS_11322]	Definition of API function ara::rds::RawDataStreamServer::ReadData
[SWS_RDS_11323]	Definition of API function ara::rds::RawDataStreamServer::ReadData
[SWS_RDS_11324]	Definition of API function ara::rds::RawDataStreamServer::WriteData
[SWS_RDS_11325]	Definition of API function ara::rds::RawDataStreamServer::WriteData
[SWS_RDS_12367]	Definition of API enum ara::rds::RawErrc
[SWS_RDS_90007]	Restrictions on using RawDataStreams
[SWS_RDS_90211]	Secure UDP and TCP channel creation for TLS and DTLS
[SWS_RDS_90212]	Using secure TLS, DTLS channels
[SWS_RDS_90213]	TLS secure channel for raw data streams using reliable transport
[SWS_RDS_90214]	DTLS secure channel for methods using unreliable transport
[SWS_RDS_90215]	IPsec secure channel between communication nodes and Transport of Raw Data Stream communication over an IPsec security association
[SWS_RDS_90216]	Socket Options configuration
[SWS_RDS_90217]	TLS properties configuration
[SWS_RDS_99004]	Ethernet endpoint configuration

▽

△

Number	Heading
[SWS_RDS_99005]	Wait for incoming connections
[SWS_RDS_99006]	Timeout handling
[SWS_RDS_99025]	Raw errors domain

Table B.1: Added Specification Items in R23-11

B.1.2 Changed Specification Items in R23-11

none

B.1.3 Deleted Specification Items in R23-11

none