Document Title	Specification of Communication Management
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	717

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Added IEEE1722 Signal-Service-Translation network binding • Fixed interoperability issues between AP and CP regarding SOME/IP Error responses • Added handling of Transport Fault Conditions • Added support for MACsec cryptograhic protection • Reworked ara::com C++ API decriptions based on header files with generated API Tables • Harmonized error codes and violations for all ara::com APIs • Editorial changes and bugfixes

▽

△

2023-11-23	R23-11	AUTOSAR Release Management	• Communication Groups are now OBSOLETE • Removed Raw Data Stream functionality from ara::com • Added new API for checking Subscription state on skeleton side • Harmonization of ara::com API Error Codes • Added clarifications for SOME/IP-SD protocol usage • Reworked usage of Access Management grants in the ara::com API • Specified lifetime requirements for event sample data • Structural changes for better overview • Editorial changes and bugfixes
2022-11-24	R22-11	AUTOSAR Release Management	• Added Static Service Connections • Added new API for ServiceStates • Clarified shutdown behavior for the Communication Management • Added specification of data types SampleType and FieldType • Added support for MACSec Secure Communication channels • Harmonization with PRS SOME/IP ServiceDiscovery Protocol document • Clarified Error codes for Fields, and E2E Error Handling • Replaced usage of SamplePtr for RawDataStreams • Editorial changes and bugfixes

▽

△

2021-11-25	R21-11	AUTOSAR Release Management	• Specified use cases and endpoint configuration for RawDataStreams • Added E2E communication protection for Fields • Added E2E profile P44m and P08m • Added new ServiceInterface element Trigger • Extend DDS Serialization of Payload chapter • Extend DDS Network binding chapter • Added Signal-Based Static Network binding • Added Freshness Value Management (FVM) • Minor vocabulary improvements and bugfixes
2020-11-30	R20-11	AUTOSAR Release Management	• Added SecOC Behavior, API and Freshness Value Management to specification • Standardized API Error Codes for ara::com API • Added unique ErrorDomain identifiers • Added Named Constructor Approach • Updated E2E Support for methods and events • Updated Raw Data Streaming chapters • Introduced optional execution context parameter to APIs with an asynchronous callback • Changed kCapabilityEnforcementError to kGrantEnforcementError • Moved magic numbers for "entry type" field to PRS_SOMEIPServiceDiscovery • Editorial Changes

▽

△

2019-11-28	R19-11	AUTOSAR Release Management	• Introduced – Signal2Service Translation Binding – Support for Invalid Values – Additional E2E support – Service Versioning – Raw Data Streaming Interface – Changed Document Status from Final to published • Minor changes and bugfixes
2019-03-29	19-03	AUTOSAR Release Management	• Predictable Resource Allocation for Samples • Usage of Future::Get/Wait with an unreliable transport • Removed exceptions on reception of malformed messages • Changes to Identity and Access Management to incorporate Grant design • Minor changes and bugfixes
2018-10-31	18-10	AUTOSAR Release Management	• Introduced Adaptive Core types • Introduced exception-less API • Refined DDS network binding • Minor changes and bugfixes
2018-03-29	18-03	AUTOSAR Release Management	• DDS Network Binding • Datatype Namespaces changed • E2E Protected Methods • Automatic Reconnection of Proxies • Minor changes and bugfixes

▽

△

2017-10-27	17-10	AUTOSAR Release Management	• Introduction of Fields • Introduction of E2E protected communication • Introduction of TLV • Improved specification of SOME/IP functional behavior • Minor changes and bugfixes
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Contents

1 Introduction and functional overview

This document contains the requirements on the functionality, API and the configuration of the AUTOSAR Adaptive Communication Management as part of the Adaptive AUTOSAR platform foundation.

The Communication Management realizes Service Oriented Communication between Adaptive AUTOSAR Applications for all levels of communication, e.g. IntraProcess, InterProcess, InterMachine. It consists of potentially generated Service Provider Skeletons and Service Requester Proxies and optionally the generic Communication Manager software for central brokering and configuration.

The Communication Management provides a built-in safety mechanism (E2E protection), which can be used for all levels of communication for events and methods.

The documentation of the Communication Management consists of two documents:

- the ARAComAPI explanatory document [1], providing explanations of the design and behavior descriptions of the ara::com API,
- this document, providing the requirements on the ara::com API.

Therefore it is recommended to read the ARAComAPI explanatory document first to get an overview and understanding, and to read this document afterward.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations that are only relevant within this specification. A general list of acronyms and abbreviations is available in [2].

Abbreviation / Acronym:	Description:
CM	Communication Management
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
SoC	Service-Oriented Communication
SecOC	Secure Onboard Communication
DTLS	Datagram Transport Layer Security
DDS	Data Distribution Service
RTPS	Real Time Publish Subscribe Protocol
TTL	Time To Live
TLV	Tag-Length-Value
RPC	Remote Procedure Call
QoS	Quality of Service
BOM	Byte Order Mark

Table 2.1: Acronyms and abbreviations used in the scope of this Document

Term:	Description:
Callable	In the context of C++ a Callable is defined as: A Callable type is a type for which the INVOKE operation (used by, e.g., std::function, std::bind, and std::thread::thread) is applicable. This operation may be performed explicitly using the library function std::invoke. (since C++17)
serializedSample	A serializedSample is the serialization of a C++ object to an array and consists of the header that is part of e2e protection and the serialized data.
Service Binding	Multi-Binding describes setups having multiple connections implemented by different technical transport layers and protocol between different instances of a single proxy or skeleton class, e.g.:
Multi-Binding	Multi-Binding describes setups having multiple connections implemented by different technical transport layers and protocol between different instances of a single proxy or skeleton class, e.g.: • A proxy class uses different transport/IPC to communicate with different skeleton instances. • Different proxy instances for the same skeleton instance uses different transport/IPC to communicate with this instance: The skeleton instance supports multiple transport mechanisms to get contacted.
Orphaned response	A received response / error message of a cancelled method call.
Cycle	A cycle describes the time interval of periodical sending and reception of messages. This is important for E2E protected messages with a message counter which shall be increased by the sending entity for every new cycle (= for every new message). Within every cycle the receiving entity shall check if a new message has arrived and if its content is usable. For more information see documents PRS_E2EProtocol (chapter 6) and SWS_E2ELibrary (chapter 9).

Table 2.2: Acronyms and abbreviations used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Explanation of ara::com API
AUTOSAR_AP_EXP_ARAComAPI

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] Specification of Adaptive Platform Core
AUTOSAR_AP_SWS_Core

[4] SOME/IP Protocol Specification
AUTOSAR_FO_PRS_SOMEIPProtocol

[5] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[6] SOME/IP Service Discovery Protocol Specification
AUTOSAR_FO_PRS_SOMEIPServiceDiscoveryProtocol

[7] E2E Protocol Specification
AUTOSAR_FO_PRS_E2EProtocol

[8] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[9] Requirements on E2E
AUTOSAR_FO_RS_E2E

[10] Requirements on Communication Management
AUTOSAR_AP_RS_CommunicationManagement

[11] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General

[12] Middleware for Real-time and Embedded Systems
http://doi.acm.org/10.1145/508448.508472

[13] Patterns, Frameworks, and Middleware: Their Synergistic Relationships
http://dl.acm.org/citation.cfm?id=776816.776917

[14] Reference Model for Service Oriented Architecture 1.0
https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

[15] Specification of Platform Types for Adaptive Platform
AUTOSAR_AP_SWS_PlatformTypes

[16] UTF-8, a transformation format of ISO 10646
http://www.ietf.org/rfc/rfc3629.txt

[17] UTF-16, an encoding of ISO 10646

http://www.ietf.org/rfc/rfc2781.txt

[18] Specification of Socket Adaptor
AUTOSAR_CP_SWS_SocketAdaptor

[19] IEEE Standard 1722-2016 - IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks

[20] Data Distribution Service (DDS), Version 1.4
http://www.omg.org/spec/DDS/1.4

[21] DDS Interoperability Wire Protocol, Version 2.2
http://www.omg.org/spec/DDSI-RTPS/2.2

[22] Extensible and Dynamic Topic Types for DDS, Version 1.2
https://www.omg.org/spec/DDS-XTypes/1.2

[23] RPC over DDS, Version 1.0
https://www.omg.org/spec/DDS-RPC/1.0

[24] The Transport Layer Security (TLS) Protocol Version 1.2
https://rfc-editor.org/rfc/rfc5246.txt

[25] Datagram Transport Layer Security Version 1.2
https://ietf.org/rfc/rfc6347.txt

[26] DDS Security, Version 1.1
https://www.omg.org/spec/DDS-SECURITY/1.1

[27] Integration of DDS Security
AUTOSAR_AP_TR_DDSSecurityIntegration

[28] Specification of Secure Onboard Communication Protocol
AUTOSAR_FO_PRS_SecOcProtocol

[29] Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode
https://ietf.org/rfc/rfc5487.txt

[30] Specification of Language Binding for modeled AP data types
AUTOSAR_AP_SWS_LanguageBindingForModeledAPdatatypes

[31] ISO/IEC 14882:2014, Information technology – Programming languages – C++
https://www.iso.org

[32] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

3.2 Further applicable specification

AUTOSAR provides a core specification [3] which is also applicable for this functional cluster. The chapter "General requirements for all Functional Clusters" of [3] shall be considered an additional and required specification for implementing this functional cluster.

4 Constraints and assumptions

4.1 Known Limitations

The current version of this document is missing some functionality which is not standardized and specified within the *SWS Communication Management* document but described in *Explanation of ara::com API* [1] and implemented in the demonstrator code:

4.1.1 Local Buffer Overruns

Currently it is not specified what happens if local buffers are full because the application accesses data slower than they are received over the network.

4.1.2 SOME/IP

The following limitations regarding the SOME/IP functionality described in [4] and [5] apply:

4.1.2.1 Optional method arguments with Tag-Length-Value serialization

[SWS_CM_CONSTR_00001] Optional method arguments with SOME/IP Tag-Length-Value serialization ⌈Communication Management does currently not support the existence of optional method arguments with the Tag-Length-Value serialization principle (described in [4] and [5].⌋

)

4.1.3 SOME/IP Service Discovery

The following limitations regarding the SOME/IP SD functionality described in [6] and [5] apply:

4.1.3.1 SOME/IP Service Discovery Discardable flag

[SWS_CM_CONSTR_00002] SOME/IP Service Discovery Discardable flag ⌈The specification does not support setting the SOME/IP Service Discovery Discardable flag of a SOME/IP entry option to 1 and reacting to the reception of an (unknown/un-

supported) option with the Discardable flag set to 1 (see [PRS_SOMEIPSD_00273], [PRS_SOMEIPSD_00275], [PRS_SOMEIPSD_00276], [PRS_SOMEIPSD_00544]).⌋

4.1.3.2 SOME/IP Service Discovery Configuration options

[SWS_CM_CONSTR_00003] SOME/IP Service Discovery Configuration options ⌈The specification does not support that SOME/IP Service Discovery configuration options (see [PRS_SOMEIPSD_00276] - [PRS_SOMEIPSD_00287]) configured via the `RequiredSomeipServiceInstance.capabilityRecord` respectively in the `ProvidedSomeipServiceInstance.capabilityRecord` and configuration options contained in a FindService respectively OfferService entry of a received SOME/IP-SD message are considered during the process of matching offered services to required services and vice versa. Any configured and/or received configuration options are simply ignored during the matching process, i.e., they don't have any impact on the result set returned by FindService() or StartFindService(). Please note that the transmission of configuration options is supported.⌋

4.1.3.3 SOME/IP Service Discovery Load balancing options

[SWS_CM_CONSTR_00004] SOME/IP Service Discovery Load balancing options ⌈The specification does not support that SOME/IP Service Discovery load balancing options configured (see [PRS_SOMEIPSD_00542], [PRS_SOMEIPSD_00544], [PRS_SOMEIPSD_00711] - [PRS_SOMEIPSD_00714]) via the `ProvidedSomeipServiceInstance.loadBalancingPriority` and `ProvidedSomeipServiceInstance.loadBalancingWeight` are included in the OfferService entry of a transmitted SOME/IP-SD message. Additionally, the specification does not support that load balancing options of an OfferService entry of a received SOME/IP-SD message are considered during the process of matching offered services to required services. Any received load balancing options are simply ignored during the matching process, i.e., they don't have any impact on the result set returned by FindService() or StartFindService().⌋

4.1.3.4 SOME/IP Service Discovery SD endpoint options

[SWS_CM_CONSTR_00005] SOME/IP Service Discovery SD endpoint options ⌈The specification does not support that IPv4/IPv6 SD endpoint options (see [PRS_SOMEIPSD_00547] - [PRS_SOMEIPSD_00552], [PRS_SOMEIPSD_00554] - [PRS_SOMEIPSD_00559], [PRS_SOMEIPSD_00650], [PRS_SOMEIPSD_00651], [PRS_-

SOMEIPSD_00654]) are included in any SOME/IP-SD entry of a transmitted SOME/IP-SD message.⌋

4.1.4 E2E Protection

The general limitations regarding E2E protection and the detectable failure modes are described in [7]. Additional, platform specific limitations regarding E2E protection are described in chapter 7.4.1.2.1 and 7.4.1.1.1.

[SWS_CM_CONSTR_00007] E2E Protection ⌈E2E protection of `ServiceInterface.trigger`s are not supported in the current version of this document.⌋

4.1.5 Timing of the network behavior

The timing of the network behavior is platform vendor specific (examples are: socket open, socket close, trigger to send a find message). This is particularly important during the Functional cluster lifecycle analysis.

4.1.6 Signal-Based IEEE1722 ACF Network binding

[SWS_CM_CONSTR_00009] No service discovery for Signal-Based IEEE1722 ACF Network binding

Status: DRAFT
Upstream requirements: RS_CM_00006

⌈Signal-Based IEEE1722 ACF Network binding does not support service discovery. All communication paths need to be configured statically.⌋

4.1.7 Thread-safety

Thread-safety for the ara::com API is not defined within this release.

5 Dependencies to other Functional Clusters

This chapter defines the dependencies of this functional cluster to other functional clusters. AUTOSAR decided not to standardize interfaces which are exclusively used between functional clusters to allow efficient implementations which might depend e.g., on the used operating system. The goal of this chapter is to provide an informative guideline for the interactions between functional clusters without specifying syntactical details. This ensures compatibility between documents specifying different functional clusters and supports parallel implementation of different functional clusters. Details of internal interfaces are up to the platform provider. Additional internal interfaces, parameters, and return values can be added. A detailed technical architecture documentation of the overall AUTOSAR Adaptive Platform is provided in [8].

5.1 Provided Interfaces

This section provides an overview of the public interfaces provided by this functional cluster towards other functional clusters.

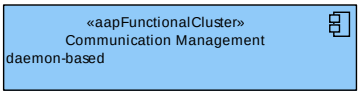

Figure 5.1: Interfaces provided by Communication Management to other Functional Clusters

Figure 5.1 shows the interfaces provided by `CommunicationManagement` to other functional clusters within the AUTOSAR Adaptive Platform. Table 5.1 lists the interfaces provided to other functional clusters within the AUTOSAR Adaptive Platform and provides a rationale.

Interface	*Functional Cluster*	*Purpose*
No provided interfaces		

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

This section provides an overview of the public interfaces required by this functional cluster from other functional clusters.

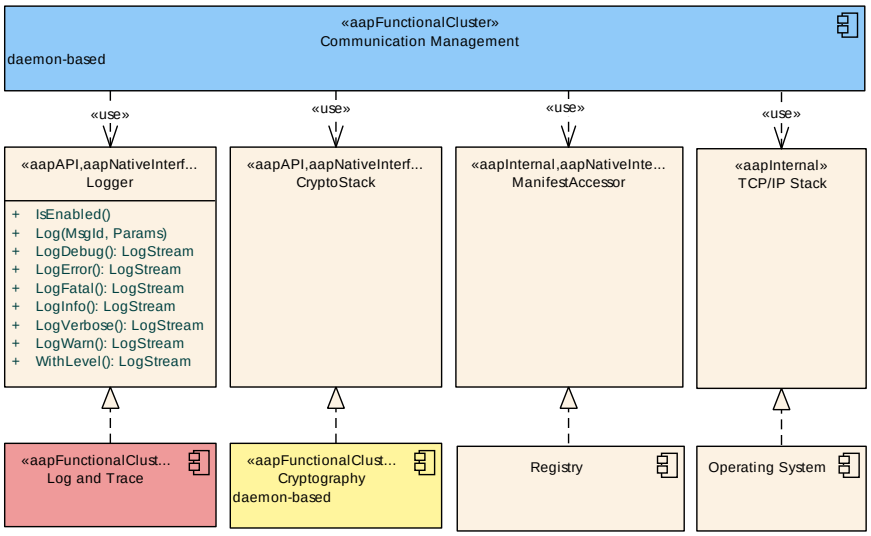

Figure 5.2: Interfaces required by Communication Management from other Functional Clusters

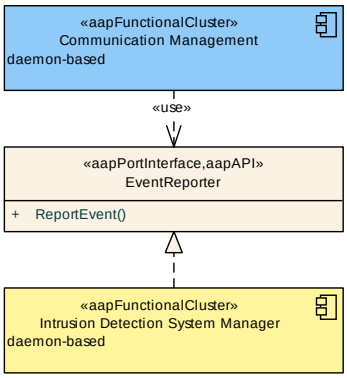

Figure 5.3: Interfaces required by Communication Management from other Functional Clusters

Figure 5.2 shows the interfaces required by `CommunicationManagement` from other functional clusters within the AUTOSAR Adaptive Platform. Table 5.2 lists the interfaces required from other functional clusters within the AUTOSAR Adaptive Platform and provides a rationale.

Functional Cluster	*Interface*	*Purpose*
Cryptography	CryptoStack	This interface may be used e.g., to establish encrypted connections and generate / verify checksums (MAC).
Intrusion Detection System Manager	EventReporter	`Communication Management` may use this interface to report security events.
Log and Trace	Logger	`Communication Management` shall use this interface to log standardized messages.
Operating System Interface	OperatingSystemInterface	`Communication Management` should use this interface to create and control `Thread`s used by the implementation.

Table 5.2: Interfaces required from other Functional Clusters

5.3 Platform dependencies

The Communication Management is dependent on the E2E protection protocol defined in [9] and [7]. The E2E functions are used to execute end-to-end communication protection between Service Provider Skeletons and Service Requester Proxies.

6 Requirements Tracing

The following tables reference the requirements specified in the Requirements on Communication Management document [10] and the AUTOSAR RS General [11], and links to the fulfilment 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
[FO_RS_Dds_00001]	DDS Compliance	[SWS_CM_09004] [SWS_CM_10431] [SWS_CM_10524] [SWS_CM_10525] [SWS_CM_10526] [SWS_CM_10527] [SWS_CM_10528] [SWS_CM_10529] [SWS_CM_10530] [SWS_CM_10531] [SWS_CM_10532] [SWS_CM_10534] [SWS_CM_10535] [SWS_CM_10536] [SWS_CM_10537] [SWS_CM_10550] [SWS_CM_11000] [SWS_CM_11001] [SWS_CM_11002] [SWS_CM_11003] [SWS_CM_11005] [SWS_CM_11006] [SWS_CM_11007] [SWS_CM_11008] [SWS_CM_11009] [SWS_CM_11010] [SWS_CM_11011] [SWS_CM_11012] [SWS_CM_11013] [SWS_CM_11014] [SWS_CM_11015] [SWS_CM_11016] [SWS_CM_11017] [SWS_CM_11018] [SWS_CM_11019] [SWS_CM_11020] [SWS_CM_11021] [SWS_CM_11022] [SWS_CM_11023] [SWS_CM_11024] [SWS_CM_11025] [SWS_CM_11026] [SWS_CM_11027] [SWS_CM_11028] [SWS_CM_11029] [SWS_CM_11030] [SWS_CM_11031] [SWS_CM_11040] [SWS_CM_11041] [SWS_CM_11042] [SWS_CM_11043] [SWS_CM_11044] [SWS_CM_11046] [SWS_CM_11047] [SWS_CM_11048] [SWS_CM_11049] [SWS_CM_11050] [SWS_CM_11100] [SWS_CM_11101] [SWS_CM_11102] [SWS_CM_11103] [SWS_CM_11104] [SWS_CM_11105] [SWS_CM_11106] [SWS_CM_11107] [SWS_CM_11108] [SWS_CM_11109] [SWS_CM_11110] [SWS_CM_11111] [SWS_CM_11112] [SWS_CM_11130] [SWS_CM_11131] [SWS_CM_11132] [SWS_CM_11133] [SWS_CM_11134] [SWS_CM_11135] [SWS_CM_11136] [SWS_CM_11137] [SWS_CM_11138] [SWS_CM_11139] [SWS_CM_11140] [SWS_CM_11141] [SWS_CM_11142] [SWS_CM_11143] [SWS_CM_11144] [SWS_CM_11145] [SWS_CM_11146] [SWS_CM_11147] [SWS_CM_11148] [SWS_CM_11149] [SWS_CM_11150] [SWS_CM_11151] [SWS_CM_11152] [SWS_CM_11153] [SWS_CM_11154] [SWS_CM_11155] [SWS_CM_11156] [SWS_CM_90218] [SWS_CM_90500] [SWS_CM_90501] [SWS_CM_90502] [SWS_CM_90503] [SWS_CM_90504] [SWS_CM_90505] [SWS_CM_90506] [SWS_CM_90507]

△

Requirement	Description	Satisfied by
		△ [SWS_CM_90508] [SWS_CM_90509] [SWS_CM_90510] [SWS_CM_90511] [SWS_CM_90512] [SWS_CM_90513] [SWS_CM_90514] [SWS_CM_90515]
[FO_RS_Dds_00002]	DDS standard serialization rules	[SWS_CM_11040] [SWS_CM_11041] [SWS_CM_11042] [SWS_CM_11043] [SWS_CM_11044] [SWS_CM_11046] [SWS_CM_11047] [SWS_CM_11048] [SWS_CM_11049] [SWS_CM_11050]
[FO_RS_Dds_00005]	DDS Quality of Service	[SWS_CM_09004] [SWS_CM_10524] [SWS_CM_10528] [SWS_CM_10550] [SWS_CM_11001] [SWS_CM_11002] [SWS_CM_11003] [SWS_CM_11005] [SWS_CM_11006] [SWS_CM_11007] [SWS_CM_11008] [SWS_CM_11009] [SWS_CM_11010] [SWS_CM_11011] [SWS_CM_11012] [SWS_CM_11013] [SWS_CM_11014] [SWS_CM_11015] [SWS_CM_11019] [SWS_CM_11100] [SWS_CM_11103] [SWS_CM_11104] [SWS_CM_11105] [SWS_CM_11106] [SWS_CM_11130] [SWS_CM_11134] [SWS_CM_11135] [SWS_CM_11144] [SWS_CM_11147] [SWS_CM_11148] [SWS_CM_11149] [SWS_CM_11150] [SWS_CM_90501] [SWS_CM_90506] [SWS_CM_90508] [SWS_CM_90511]
[FO_RS_Dds_00007]	Type Definition	[SWS_CM_10431] [SWS_CM_10524] [SWS_CM_10525] [SWS_CM_11015] [SWS_CM_11016] [SWS_CM_11041] [SWS_CM_11042] [SWS_CM_11043] [SWS_CM_11044] [SWS_CM_11046] [SWS_CM_11047] [SWS_CM_11048] [SWS_CM_11049] [SWS_CM_11050] [SWS_CM_11101] [SWS_CM_11102] [SWS_CM_11131] [SWS_CM_11145] [SWS_CM_11146] [SWS_CM_90508]
[FO_RS_Dds_00008]	Customization	[SWS_CM_09004] [SWS_CM_10524] [SWS_CM_10526] [SWS_CM_10527] [SWS_CM_10528] [SWS_CM_10529] [SWS_CM_10530] [SWS_CM_10531] [SWS_CM_10532] [SWS_CM_10534] [SWS_CM_10535] [SWS_CM_10536] [SWS_CM_10537] [SWS_CM_11005] [SWS_CM_11006] [SWS_CM_11007] [SWS_CM_11008] [SWS_CM_11009] [SWS_CM_11010] [SWS_CM_11011] [SWS_CM_11012] [SWS_CM_11013] [SWS_CM_11014] [SWS_CM_11015] [SWS_CM_11019] [SWS_CM_11020] [SWS_CM_11021] [SWS_CM_11022] [SWS_CM_11023] [SWS_CM_11024] [SWS_CM_11025] [SWS_CM_11026] [SWS_CM_11027] [SWS_CM_11028] [SWS_CM_11030] [SWS_CM_11031] [SWS_CM_11100] [SWS_CM_11103] [SWS_CM_11104] [SWS_CM_11105] [SWS_CM_11106] [SWS_CM_11107] [SWS_CM_11108] [SWS_CM_11109] [SWS_CM_11110] [SWS_CM_11111] [SWS_CM_11112] [SWS_CM_11130] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_11132] [SWS_CM_11133] [SWS_CM_11134] [SWS_CM_11135] [SWS_CM_11136] [SWS_CM_11137] [SWS_CM_11138] [SWS_CM_11139] [SWS_CM_11140] [SWS_CM_11141] [SWS_CM_11142] [SWS_CM_11143] [SWS_CM_11144] [SWS_CM_11147] [SWS_CM_11148] [SWS_CM_11149] [SWS_CM_11150] [SWS_CM_11151] [SWS_CM_11152] [SWS_CM_11153] [SWS_CM_11154] [SWS_CM_11155] [SWS_CM_11156] [SWS_CM_90500] [SWS_CM_90502] [SWS_CM_90503] [SWS_CM_90504] [SWS_CM_90505] [SWS_CM_90506] [SWS_CM_90507] [SWS_CM_90508] [SWS_CM_90509] [SWS_CM_90510] [SWS_CM_90511] [SWS_CM_90512] [SWS_CM_90513] [SWS_CM_90514] [SWS_CM_90515]
[FO_RS_Dds_00009]	Security mechanism	[SWS_CM_90218]
[FO_RS_Dds_00015]	Publish	[SWS_CM_10526] [SWS_CM_10550] [SWS_CM_11003] [SWS_CM_11005] [SWS_CM_11017] [SWS_CM_11029] [SWS_CM_11030] [SWS_CM_11031] [SWS_CM_11103] [SWS_CM_11106] [SWS_CM_11107] [SWS_CM_11108] [SWS_CM_11112] [SWS_CM_11132] [SWS_CM_11151] [SWS_CM_11152] [SWS_CM_11156] [SWS_CM_90503] [SWS_CM_90504] [SWS_CM_90505] [SWS_CM_90506] [SWS_CM_90507] [SWS_CM_90508]
[FO_RS_Dds_00016]	Subscribe	[SWS_CM_10527] [SWS_CM_10528] [SWS_CM_10529] [SWS_CM_10536] [SWS_CM_10537] [SWS_CM_11005] [SWS_CM_11018] [SWS_CM_11019] [SWS_CM_11020] [SWS_CM_11031] [SWS_CM_11104] [SWS_CM_11105] [SWS_CM_11109] [SWS_CM_11110] [SWS_CM_11111] [SWS_CM_11133] [SWS_CM_11134] [SWS_CM_11135] [SWS_CM_11153] [SWS_CM_90503] [SWS_CM_90505] [SWS_CM_90507]
[FO_RS_MACsec_-00001]	MACsec Protocol support	[SWS_CM_99040]
[FO_RS_MACsec_-00006]	MACsec support for Adaptive AUTOSAR Platform	[SWS_CM_99040]

▽

△

Requirement	Description	Satisfied by
[RS_AP_00114]	C++ interface shall be compatible with C++14	[SWS_CM_00002] [SWS_CM_00003] [SWS_CM_00004] [SWS_CM_00006] [SWS_CM_00007] [SWS_CM_00008] [SWS_CM_00048] [SWS_CM_00049] [SWS_CM_00053] [SWS_CM_00054] [SWS_CM_00055] [SWS_CM_00056] [SWS_CM_00101] [SWS_CM_00111] [SWS_CM_00112] [SWS_CM_00113] [SWS_CM_00114] [SWS_CM_00116] [SWS_CM_00118] [SWS_CM_00119] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00125] [SWS_CM_00130] [SWS_CM_00131] [SWS_CM_00134] [SWS_CM_00135] [SWS_CM_00136] [SWS_CM_00137] [SWS_CM_00152] [SWS_CM_00153] [SWS_CM_00162] [SWS_CM_00181] [SWS_CM_00191] [SWS_CM_00192] [SWS_CM_00193] [SWS_CM_00194] [SWS_CM_00195] [SWS_CM_00196] [SWS_CM_00197] [SWS_CM_00199] [SWS_CM_00226] [SWS_CM_00250] [SWS_CM_00302] [SWS_CM_00306] [SWS_CM_00309] [SWS_CM_00312] [SWS_CM_00317] [SWS_CM_00318] [SWS_CM_00319] [SWS_CM_00334] [SWS_CM_00351] [SWS_CM_00352] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_00721] [SWS_CM_00723] [SWS_CM_00726] [SWS_CM_00727] [SWS_CM_00810] [SWS_CM_01002] [SWS_CM_01005] [SWS_CM_01006] [SWS_CM_01009] [SWS_CM_01010] [SWS_CM_01012] [SWS_CM_01013] [SWS_CM_01015] [SWS_CM_01018] [SWS_CM_01031] [SWS_CM_10383] [SWS_CM_10438] [SWS_CM_10440] [SWS_CM_10446] [SWS_CM_11352] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11360] [SWS_CM_11362] [SWS_CM_11365] [SWS_CM_11371] [SWS_CM_11377] [SWS_CM_11379] [SWS_CM_11400] [SWS_CM_11402] [SWS_CM_11403] [SWS_CM_11500] [SWS_CM_11501] [SWS_CM_11502] [SWS_CM_11503] [SWS_CM_11520] [SWS_CM_11521] [SWS_CM_11522] [SWS_CM_11523] [SWS_CM_11524] [SWS_CM_11525] [SWS_CM_11532] [SWS_CM_11533] [SWS_CM_11534] [SWS_CM_11535] [SWS_CM_11536] [SWS_CM_11537] [SWS_CM_11538] [SWS_CM_11539] [SWS_CM_11540] [SWS_CM_11541] [SWS_CM_11542] [SWS_CM_11543] [SWS_CM_11544] [SWS_CM_11545] [SWS_CM_11546] [SWS_CM_11547] [SWS_CM_11548] [SWS_CM_11549] [SWS_CM_11550] [SWS_CM_11551] [SWS_CM_11552] [SWS_CM_11603] [SWS_CM_11605] [SWS_CM_11608] [SWS_CM_11609] [SWS_CM_11610] [SWS_CM_11611] [SWS_CM_11615] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_12002] [SWS_CM_12003] [SWS_CM_12021] [SWS_CM_12022] [SWS_CM_12026] [SWS_CM_90420] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90434] [SWS_CM_90435] [SWS_CM_90437] [SWS_CM_90438] [SWS_CM_98444] [SWS_CM_99332] [SWS_CM_99333] [SWS_CM_99444] [SWS_CM_99445] [SWS_CM_99446] [SWS_CM_99447] [SWS_CM_99556] [SWS_CM_99557] [SWS_CM_99558] [SWS_CM_99559]
[RS_AP_00115]	Public namespaces	[SWS_CM_00118] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00152] [SWS_CM_00153] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_11352] [SWS_CM_11365] [SWS_CM_12021] [SWS_CM_12022] [SWS_CM_12026] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90438]
[RS_AP_00116]	Header file name	[SWS_CM_01002] [SWS_CM_01012] [SWS_CM_01013] [SWS_CM_11379] [SWS_CM_11503]
[RS_AP_00119]	Return values / application errors	[SWS_CM_00048] [SWS_CM_00049] [SWS_CM_00118] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_10383] [SWS_CM_10440] [SWS_CM_11265] [SWS_CM_11352] [SWS_CM_11365] [SWS_CM_12021] [SWS_CM_12022] [SWS_CM_12026] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_99030]
[RS_AP_00120]	Method and Function names	[SWS_CM_00101] [SWS_CM_00111] [SWS_CM_00112] [SWS_CM_00113] [SWS_CM_00114] [SWS_CM_00116] [SWS_CM_00118] [SWS_CM_00119] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00125] [SWS_CM_00181] [SWS_CM_00192] [SWS_CM_00195] [SWS_CM_00196] [SWS_CM_00199] [SWS_CM_00226] [SWS_CM_00250] [SWS_CM_00309] [SWS_CM_00334] [SWS_CM_00351] [SWS_CM_00352] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_00723] [SWS_CM_00810] [SWS_CM_11328] [SWS_CM_11330] [SWS_CM_11331] [SWS_CM_11332] [SWS_CM_11333] [SWS_CM_11334] [SWS_CM_11335] [SWS_CM_11336] [SWS_CM_11337] [SWS_CM_11352] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11360] [SWS_CM_11362] [SWS_CM_11365] [SWS_CM_11603] [SWS_CM_11605] [SWS_CM_11608] [SWS_CM_11609] [SWS_CM_11610] [SWS_CM_11611] [SWS_CM_11615] [SWS_CM_12002] [SWS_CM_12003] [SWS_CM_12502] [SWS_CM_12504] [SWS_CM_12505] [SWS_CM_12506] [SWS_CM_12507] [SWS_CM_12508] [SWS_CM_12509] [SWS_CM_12510] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_12511] [SWS_CM_12513] [SWS_CM_12515] [SWS_CM_12516] [SWS_CM_12517] [SWS_CM_12518] [SWS_CM_12519] [SWS_CM_12520] [SWS_CM_12521] [SWS_CM_12522] [SWS_CM_90435] [SWS_CM_90438] [SWS_CM_99333]
[RS_AP_00121]	Parameter names	[SWS_CM_00113] [SWS_CM_00118] [SWS_CM_00119] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00125] [SWS_CM_00130] [SWS_CM_00131] [SWS_CM_00152] [SWS_CM_00153] [SWS_CM_00162] [SWS_CM_00181] [SWS_CM_00226] [SWS_CM_00250] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00701] [SWS_CM_00721] [SWS_CM_10438] [SWS_CM_11328] [SWS_CM_11332] [SWS_CM_11333] [SWS_CM_11335] [SWS_CM_11352] [SWS_CM_11365] [SWS_CM_11605] [SWS_CM_11610] [SWS_CM_12502] [SWS_CM_12508] [SWS_CM_12509] [SWS_CM_12511] [SWS_CM_12513] [SWS_CM_12519] [SWS_CM_12520] [SWS_CM_12522] [SWS_CM_90437]
[RS_AP_00122]	Type names	[SWS_CM_00002] [SWS_CM_00004] [SWS_CM_00306] [SWS_CM_00312] [SWS_CM_00319] [SWS_CM_10432] [SWS_CM_11327] [SWS_CM_11329] [SWS_CM_11534] [SWS_CM_11535] [SWS_CM_11536] [SWS_CM_11537] [SWS_CM_11538] [SWS_CM_11539] [SWS_CM_11540] [SWS_CM_11541] [SWS_CM_11542] [SWS_CM_11543] [SWS_CM_11544] [SWS_CM_11545] [SWS_CM_11546] [SWS_CM_11547] [SWS_CM_12501] [SWS_CM_12503] [SWS_CM_12512] [SWS_CM_12514] [SWS_CM_90420] [SWS_CM_99030]
[RS_AP_00127]	Usage of ara::core types	[SWS_CM_00048] [SWS_CM_00049] [SWS_CM_00053] [SWS_CM_00054] [SWS_CM_00055] [SWS_CM_00056] [SWS_CM_00112] [SWS_CM_00113] [SWS_CM_00114] [SWS_CM_00116] [SWS_CM_00118] [SWS_CM_00152] [SWS_CM_00191] [SWS_CM_00192] [SWS_CM_00193] [SWS_CM_00194] [SWS_CM_00195] [SWS_CM_00196] [SWS_CM_00197] [SWS_CM_00199] [SWS_CM_00226] [SWS_CM_00302] [SWS_CM_00352] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_10432] [SWS_CM_10438] [SWS_CM_10440] [SWS_CM_10446] [SWS_CM_11327] [SWS_CM_11329] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11360] [SWS_CM_11362] [SWS_CM_11520] [SWS_CM_11521] [SWS_CM_11522] [SWS_CM_11523] [SWS_CM_11524] [SWS_CM_11525] [SWS_CM_11550] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_11603] [SWS_CM_11608] [SWS_CM_11610] [SWS_CM_11611] [SWS_CM_12501] [SWS_CM_12503] [SWS_CM_12512] [SWS_CM_12514] [SWS_CM_12515] [SWS_CM_12516] [SWS_CM_12517] [SWS_CM_12518] [SWS_CM_99333] [SWS_CM_99556]
[RS_AP_00128]	Error reporting	[SWS_CM_00112] [SWS_CM_00113] [SWS_CM_00114] [SWS_CM_00116] [SWS_CM_00191] [SWS_CM_00192] [SWS_CM_00195] [SWS_CM_00196] [SWS_CM_00199] [SWS_CM_00226] [SWS_CM_00352] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_10438] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11360] [SWS_CM_11362] [SWS_CM_11550] [SWS_CM_11603] [SWS_CM_11608] [SWS_CM_11610] [SWS_CM_11611] [SWS_CM_99333] [SWS_CM_99556]
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment	[SWS_CM_10432] [SWS_CM_10474] [SWS_CM_11327] [SWS_CM_11328] [SWS_CM_11329] [SWS_CM_11330] [SWS_CM_11331] [SWS_CM_11332] [SWS_CM_11333] [SWS_CM_11334] [SWS_CM_11335] [SWS_CM_11336] [SWS_CM_11337] [SWS_CM_11342] [SWS_CM_12501] [SWS_CM_12502] [SWS_CM_12503] [SWS_CM_12504] [SWS_CM_12505] [SWS_CM_12506] [SWS_CM_12507] [SWS_CM_12508] [SWS_CM_12509] [SWS_CM_12510] [SWS_CM_12511] [SWS_CM_12512] [SWS_CM_12513] [SWS_CM_12514] [SWS_CM_12515] [SWS_CM_12516] [SWS_CM_12517] [SWS_CM_12518] [SWS_CM_12519] [SWS_CM_12520] [SWS_CM_12521] [SWS_CM_12522]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_CM_00306] [SWS_CM_00705] [SWS_CM_10438] [SWS_CM_11328] [SWS_CM_11330] [SWS_CM_11331] [SWS_CM_11332] [SWS_CM_11334] [SWS_CM_11335] [SWS_CM_11336] [SWS_CM_11337] [SWS_CM_11371] [SWS_CM_11534] [SWS_CM_11535] [SWS_CM_11536] [SWS_CM_11537] [SWS_CM_11538] [SWS_CM_11539] [SWS_CM_11540] [SWS_CM_11541] [SWS_CM_11542] [SWS_CM_11543] [SWS_CM_11544] [SWS_CM_11545] [SWS_CM_11546] [SWS_CM_11547] [SWS_CM_11603] [SWS_CM_12502] [SWS_CM_12504] [SWS_CM_12505] [SWS_CM_12506] [SWS_CM_12507] [SWS_CM_12508] [SWS_CM_12510] [SWS_CM_12511] [SWS_CM_12513] [SWS_CM_12519] [SWS_CM_12521] [SWS_CM_12522] [SWS_CM_90420]

▽

△

Requirement	Description	Satisfied by
[RS_AP_00135]	Avoidance of shared ownership	[SWS_CM_00306] [SWS_CM_11534] [SWS_CM_11535] [SWS_CM_11536] [SWS_CM_11537] [SWS_CM_11538] [SWS_CM_11539] [SWS_CM_11540] [SWS_CM_11541] [SWS_CM_11542] [SWS_CM_11543] [SWS_CM_11544] [SWS_CM_11545] [SWS_CM_11546] [SWS_CM_11547] [SWS_CM_90420]
[RS_AP_00136]	Usage of string types	[SWS_CM_10054] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10285] [SWS_CM_11046]
[RS_AP_00137]	Connecting run-time interface with model	[SWS_CM_00118] [SWS_CM_00152] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_10450] [SWS_CM_10452] [SWS_CM_10590]
[RS_AP_00138]	Return type of asynchronous function calls	[SWS_CM_00112] [SWS_CM_00113] [SWS_CM_00114] [SWS_CM_00116] [SWS_CM_00191] [SWS_CM_00192] [SWS_CM_00195] [SWS_CM_00196] [SWS_CM_00197] [SWS_CM_00199] [SWS_CM_00352] [SWS_CM_10414] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11360] [SWS_CM_11362] [SWS_CM_11550] [SWS_CM_11608] [SWS_CM_11611] [SWS_CM_99333] [SWS_CM_99556]
[RS_AP_00139]	Return type of synchronous function calls	[SWS_CM_00195] [SWS_CM_00226] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_10438] [SWS_CM_11603] [SWS_CM_11610]
[RS_AP_00145]	Availability of special member functions	[SWS_CM_00306] [SWS_CM_00317] [SWS_CM_00318] [SWS_CM_10446] [SWS_CM_11370] [SWS_CM_11371] [SWS_CM_11532] [SWS_CM_11533] [SWS_CM_11534] [SWS_CM_11535] [SWS_CM_11536] [SWS_CM_11537] [SWS_CM_11538] [SWS_CM_11539] [SWS_CM_11540] [SWS_CM_11541] [SWS_CM_11542] [SWS_CM_11543] [SWS_CM_11544] [SWS_CM_11545] [SWS_CM_11546] [SWS_CM_11547] [SWS_CM_90420]
[RS_AP_00146]	Classes whose construction requires interaction by the ARA framework	[SWS_CM_00349] [SWS_CM_00353]
[RS_AP_00151]	C++ Core Guidelines	[SWS_CM_11608] [SWS_CM_11611]
[RS_CM_00001]	The Communication Management shall provide a standardized header file structure for each service.	[SWS_CM_01002] [SWS_CM_01012] [SWS_CM_01013] [SWS_CM_10453] [SWS_CM_11379] [SWS_CM_11503]
[RS_CM_00002]	The service header files shall define the namespace for the respective service.	[SWS_CM_00003] [SWS_CM_01005] [SWS_CM_01006] [SWS_CM_01007] [SWS_CM_01009] [SWS_CM_01010] [SWS_CM_01015] [SWS_CM_01018] [SWS_CM_01031] [SWS_CM_11377] [SWS_CM_11500] [SWS_CM_11501] [SWS_CM_11502] [SWS_CM_11504] [SWS_CM_11505] [SWS_CM_98444] [SWS_CM_98447]

▽

△

Requirement	Description	Satisfied by
[RS_CM_00004]	Communication Management shall support the translation between signal-based and service-oriented communication	[SWS_CM_00057] [SWS_CM_00058] [SWS_CM_00059] [SWS_CM_00060] [SWS_CM_00061] [SWS_CM_00062] [SWS_CM_00063] [SWS_CM_00064] [SWS_CM_00065] [SWS_CM_00066] [SWS_CM_00067] [SWS_CM_00068] [SWS_CM_00069] [SWS_CM_00070] [SWS_CM_00071] [SWS_CM_00072] [SWS_CM_00073] [SWS_CM_00074] [SWS_CM_00075] [SWS_CM_00076] [SWS_CM_00077] [SWS_CM_00078] [SWS_CM_00079] [SWS_CM_00080] [SWS_CM_10363] [SWS_CM_10517] [SWS_CM_10518] [SWS_CM_10519] [SWS_CM_10520] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_10523] [SWS_CM_80001] [SWS_CM_80003] [SWS_CM_80004] [SWS_CM_80017] [SWS_CM_80019] [SWS_CM_80020] [SWS_CM_80021] [SWS_CM_80022] [SWS_CM_80023] [SWS_CM_80024] [SWS_CM_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80030] [SWS_CM_80032] [SWS_CM_80033] [SWS_CM_80063] [SWS_CM_80064] [SWS_CM_80065] [SWS_CM_80066] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069] [SWS_CM_80070] [SWS_CM_80072] [SWS_CM_80074] [SWS_CM_80075] [SWS_CM_80100] [SWS_CM_80101] [SWS_CM_80102] [SWS_CM_80103] [SWS_CM_80104] [SWS_CM_80501] [SWS_CM_80502] [SWS_CM_80503] [SWS_CM_80504] [SWS_CM_80505] [SWS_CM_80506] [SWS_CM_80507] [SWS_CM_80508] [SWS_CM_80509] [SWS_CM_80510] [SWS_CM_80511] [SWS_CM_80512] [SWS_CM_80513] [SWS_CM_80514] [SWS_CM_80515]
[RS_CM_00005]	Handling of malformed messages or with errors	[SWS_CM_10416]
[RS_CM_00006]	Support of IEEE1722 Stream ACF Message handling	[SWS_CM_00057] [SWS_CM_00058] [SWS_CM_00059] [SWS_CM_00060] [SWS_CM_00061] [SWS_CM_00062] [SWS_CM_00063] [SWS_CM_00064] [SWS_CM_00065] [SWS_CM_00066] [SWS_CM_00067] [SWS_CM_00068] [SWS_CM_00069] [SWS_CM_00070] [SWS_CM_00071] [SWS_CM_00072] [SWS_CM_00073] [SWS_CM_00074] [SWS_CM_00075] [SWS_CM_00076] [SWS_CM_00077] [SWS_CM_00078] [SWS_CM_00079] [SWS_CM_00080] [SWS_CM_00081] [SWS_CM_00082] [SWS_CM_CONSTR_00009]

▽

△

Requirement	Description	Satisfied by
[RS_CM_00101]	Communication Management shall provide an interface to offer services	[SWS_CM_00002] [SWS_CM_00053] [SWS_CM_00054] [SWS_CM_00055] [SWS_CM_00056] [SWS_CM_00101] [SWS_CM_00102] [SWS_CM_00103] [SWS_CM_00104] [SWS_CM_00130] [SWS_CM_00134] [SWS_CM_00135] [SWS_CM_00152] [SWS_CM_00153] [SWS_CM_00201] [SWS_CM_00203] [SWS_CM_00302] [SWS_CM_00319] [SWS_CM_09004] [SWS_CM_10410] [SWS_CM_10450] [SWS_CM_10451] [SWS_CM_10458] [SWS_CM_10550] [SWS_CM_11001] [SWS_CM_11002] [SWS_CM_11003] [SWS_CM_11029] [SWS_CM_11030] [SWS_CM_11031] [SWS_CM_11520] [SWS_CM_11521] [SWS_CM_11522] [SWS_CM_11523] [SWS_CM_11524] [SWS_CM_11525] [SWS_CM_11548] [SWS_CM_11549] [SWS_CM_12019] [SWS_CM_90500] [SWS_CM_90502] [SWS_CM_90503] [SWS_CM_90504] [SWS_CM_90505] [SWS_CM_90506] [SWS_CM_90507] [SWS_CM_90508]
[RS_CM_00102]	Communication Management shall provide an interface to find services	[SWS_CM_00004] [SWS_CM_00053] [SWS_CM_00054] [SWS_CM_00055] [SWS_CM_00056] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00124] [SWS_CM_00125] [SWS_CM_00131] [SWS_CM_00136] [SWS_CM_00137] [SWS_CM_00202] [SWS_CM_00209] [SWS_CM_00302] [SWS_CM_00303] [SWS_CM_00304] [SWS_CM_00312] [SWS_CM_00317] [SWS_CM_00318] [SWS_CM_00319] [SWS_CM_00383] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_10382] [SWS_CM_10438] [SWS_CM_10446] [SWS_CM_10491] [SWS_CM_11006] [SWS_CM_11007] [SWS_CM_11008] [SWS_CM_11009] [SWS_CM_11010] [SWS_CM_11011] [SWS_CM_11012] [SWS_CM_11041] [SWS_CM_11352] [SWS_CM_11365] [SWS_CM_11520] [SWS_CM_11521] [SWS_CM_11522] [SWS_CM_11523] [SWS_CM_11524] [SWS_CM_11525] [SWS_CM_11532] [SWS_CM_11533] [SWS_CM_11551] [SWS_CM_11552] [SWS_CM_90500] [SWS_CM_90510] [SWS_CM_90511] [SWS_CM_90512] [SWS_CM_90513] [SWS_CM_90514] [SWS_CM_99030] [SWS_CM_99444]

▽

△

Requirement	Description	Satisfied by
[RS_CM_00103]	Communication Management shall provide an interface to subscribe to a specific event provided by an instance of a certain service	[SWS_CM_00005] [SWS_CM_00051] [SWS_CM_00141] [SWS_CM_00205] [SWS_CM_00310] [SWS_CM_00311] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_00700] [SWS_CM_00723] [SWS_CM_00727] [SWS_CM_00810] [SWS_CM_10377] [SWS_CM_10381] [SWS_CM_10527] [SWS_CM_10528] [SWS_CM_10529] [SWS_CM_11018] [SWS_CM_11019] [SWS_CM_11020] [SWS_CM_11133] [SWS_CM_11134] [SWS_CM_11135] [SWS_CM_11401] [SWS_CM_11601] [SWS_CM_12002] [SWS_CM_12003]
[RS_CM_00104]	Communication Management shall provide an interface to stop the subscription to an event of a service instance	[SWS_CM_00151] [SWS_CM_00207] [SWS_CM_00310] [SWS_CM_00311] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_10378] [SWS_CM_10530] [SWS_CM_11021] [SWS_CM_11136] [SWS_CM_11602]
[RS_CM_00105]	Communication Management shall provide an interface to stop offering services	[SWS_CM_00111] [SWS_CM_00204] [SWS_CM_11005] [SWS_CM_90509]
[RS_CM_00106]	Communication Management shall provide a means to monitor the state of the subscription to an event	[SWS_CM_00310] [SWS_CM_00311] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_00316] [SWS_CM_00333] [SWS_CM_00334] [SWS_CM_10531] [SWS_CM_10536] [SWS_CM_10537] [SWS_CM_11022] [SWS_CM_11027] [SWS_CM_11028] [SWS_CM_11137] [SWS_CM_11142] [SWS_CM_11143] [SWS_CM_11604] [SWS_CM_11607] [SWS_CM_11609] [SWS_CM_12006] [SWS_CM_99035]
[RS_CM_00107]	Communication Management shall provide a means to automatically update a proxy instance in case of restart of the offered service	[SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_10383] [SWS_CM_10491] [SWS_CM_12006]
[RS_CM_00108]	Service Communication – Uniqueness of offered service	[SWS_CM_00102]
[RS_CM_00200]	The Communication Management shall transform Fully Qualified Service IDs to communication protocol specific Service IDs	[SWS_CM_00051] [SWS_CM_00079] [SWS_CM_00102] [SWS_CM_00118] [SWS_CM_00202] [SWS_CM_00203] [SWS_CM_00205] [SWS_CM_09004] [SWS_CM_10291] [SWS_CM_10292] [SWS_CM_10293] [SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10303] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10314] [SWS_CM_10323] [SWS_CM_10325] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10346] [SWS_CM_10377] [SWS_CM_10381] [SWS_CM_10452] [SWS_CM_10512] [SWS_CM_10513] [SWS_CM_10514] [SWS_CM_10519] [SWS_CM_10520] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_10550] [SWS_CM_10590] [SWS_CM_11001] [SWS_CM_11002] [SWS_CM_11003] [SWS_CM_11006] [SWS_CM_11007] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_11008] [SWS_CM_11009] [SWS_CM_11010] [SWS_CM_11011] [SWS_CM_11012] [SWS_CM_11013] [SWS_CM_11014] [SWS_CM_11029] [SWS_CM_11030] [SWS_CM_11031] [SWS_CM_11041] [SWS_CM_11101] [SWS_CM_11102] [SWS_CM_11107] [SWS_CM_11112] [SWS_CM_11151] [SWS_CM_11506] [SWS_CM_11507] [SWS_CM_11508] [SWS_CM_11509] [SWS_CM_11510] [SWS_CM_11511] [SWS_CM_11512] [SWS_CM_11513] [SWS_CM_11514] [SWS_CM_11515] [SWS_CM_11516] [SWS_CM_11517] [SWS_CM_11518] [SWS_CM_11519] [SWS_CM_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_90502] [SWS_CM_90503] [SWS_CM_90504] [SWS_CM_90505] [SWS_CM_90506] [SWS_CM_90507] [SWS_CM_90508] [SWS_CM_90510] [SWS_CM_90511] [SWS_CM_90512] [SWS_CM_90513] [SWS_CM_90514] [SWS_CM_90515]
[RS_CM_00201]	Communication Management shall provide an API to send events to other applications	[SWS_CM_00052] [SWS_CM_00076] [SWS_CM_00077] [SWS_CM_00078] [SWS_CM_00079] [SWS_CM_00162] [SWS_CM_00252] [SWS_CM_00253] [SWS_CM_00254] [SWS_CM_00255] [SWS_CM_00256] [SWS_CM_00257] [SWS_CM_00258] [SWS_CM_00259] [SWS_CM_00260] [SWS_CM_00264] [SWS_CM_00265] [SWS_CM_00308] [SWS_CM_00721] [SWS_CM_00726] [SWS_CM_10034] [SWS_CM_10036] [SWS_CM_10037] [SWS_CM_10042] [SWS_CM_10053] [SWS_CM_10054] [SWS_CM_10055] [SWS_CM_10056] [SWS_CM_10057] [SWS_CM_10058] [SWS_CM_10059] [SWS_CM_10060] [SWS_CM_10070] [SWS_CM_10072] [SWS_CM_10076] [SWS_CM_10088] [SWS_CM_10098] [SWS_CM_10099] [SWS_CM_10218] [SWS_CM_10219] [SWS_CM_10222] [SWS_CM_10227] [SWS_CM_10230] [SWS_CM_10234] [SWS_CM_10235] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10248] [SWS_CM_10251] [SWS_CM_10252] [SWS_CM_10253] [SWS_CM_10254] [SWS_CM_10255] [SWS_CM_10256] [SWS_CM_10257] [SWS_CM_10258] [SWS_CM_10259] [SWS_CM_10260] [SWS_CM_10261] [SWS_CM_10262] [SWS_CM_10263] [SWS_CM_10264] [SWS_CM_10265] [SWS_CM_10266] [SWS_CM_10267] [SWS_CM_10268] [SWS_CM_10269] [SWS_CM_10270] [SWS_CM_10271] [SWS_CM_10272] [SWS_CM_10273] [SWS_CM_10274] [SWS_CM_10275] [SWS_CM_10276] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_10277] [SWS_CM_10278] [SWS_CM_10279] [SWS_CM_10280] [SWS_CM_10281] [SWS_CM_10282] [SWS_CM_10283] [SWS_CM_10284] [SWS_CM_10285] [SWS_CM_10287] [SWS_CM_10288] [SWS_CM_10289] [SWS_CM_10290] [SWS_CM_10291] [SWS_CM_10292] [SWS_CM_10293] [SWS_CM_10294] [SWS_CM_10319] [SWS_CM_10320] [SWS_CM_10321] [SWS_CM_10322] [SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10325] [SWS_CM_10326] [SWS_CM_10361] [SWS_CM_10363] [SWS_CM_10459] [SWS_CM_10511] [SWS_CM_10512] [SWS_CM_10513] [SWS_CM_10514] [SWS_CM_10517] [SWS_CM_10518] [SWS_CM_10519] [SWS_CM_10520] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_10524] [SWS_CM_10525] [SWS_CM_10526] [SWS_CM_11015] [SWS_CM_11016] [SWS_CM_11017] [SWS_CM_11040] [SWS_CM_11042] [SWS_CM_11043] [SWS_CM_11044] [SWS_CM_11046] [SWS_CM_11047] [SWS_CM_11048] [SWS_CM_11049] [SWS_CM_11050] [SWS_CM_11130] [SWS_CM_11131] [SWS_CM_11132] [SWS_CM_11262] [SWS_CM_11263] [SWS_CM_11400] [SWS_CM_80021] [SWS_CM_80022] [SWS_CM_80023] [SWS_CM_80024] [SWS_CM_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80032] [SWS_CM_80063] [SWS_CM_80064] [SWS_CM_80065] [SWS_CM_80066] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069] [SWS_CM_80074] [SWS_CM_90437] [SWS_CM_90438] [SWS_CM_90501] [SWS_CM_99031] [SWS_CM_99032] [SWS_CM_99033] [SWS_CM_99445] [SWS_CM_99557] [SWS_CM_99559]
[RS_CM_00202]	Communication Management shall provide an API to the application to poll received events	[SWS_CM_00052] [SWS_CM_00081] [SWS_CM_00082] [SWS_CM_00226] [SWS_CM_00227] [SWS_CM_00252] [SWS_CM_00253] [SWS_CM_00254] [SWS_CM_00255] [SWS_CM_00256] [SWS_CM_00257] [SWS_CM_00258] [SWS_CM_00259] [SWS_CM_00260] [SWS_CM_00264] [SWS_CM_00265] [SWS_CM_00306] [SWS_CM_00701] [SWS_CM_00703] [SWS_CM_00705] [SWS_CM_00707] [SWS_CM_10016] [SWS_CM_10017] [SWS_CM_10036] [SWS_CM_10037] [SWS_CM_10042] [SWS_CM_10053] [SWS_CM_10054] [SWS_CM_10055] [SWS_CM_10056] [SWS_CM_10057] [SWS_CM_10058] [SWS_CM_10059] [SWS_CM_10060] [SWS_CM_10070] [SWS_CM_10072] [SWS_CM_10076] [SWS_CM_10088] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_10098] [SWS_CM_10099] [SWS_CM_10169] [SWS_CM_10218] [SWS_CM_10219] [SWS_CM_10222] [SWS_CM_10227] [SWS_CM_10230] [SWS_CM_10234] [SWS_CM_10235] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10248] [SWS_CM_10251] [SWS_CM_10252] [SWS_CM_10253] [SWS_CM_10254] [SWS_CM_10255] [SWS_CM_10256] [SWS_CM_10257] [SWS_CM_10258] [SWS_CM_10259] [SWS_CM_10260] [SWS_CM_10261] [SWS_CM_10262] [SWS_CM_10264] [SWS_CM_10265] [SWS_CM_10266] [SWS_CM_10267] [SWS_CM_10268] [SWS_CM_10269] [SWS_CM_10270] [SWS_CM_10271] [SWS_CM_10272] [SWS_CM_10273] [SWS_CM_10274] [SWS_CM_10275] [SWS_CM_10276] [SWS_CM_10277] [SWS_CM_10278] [SWS_CM_10279] [SWS_CM_10280] [SWS_CM_10281] [SWS_CM_10282] [SWS_CM_10283] [SWS_CM_10284] [SWS_CM_10285] [SWS_CM_10295] [SWS_CM_10327] [SWS_CM_10361] [SWS_CM_10459] [SWS_CM_10532] [SWS_CM_11023] [SWS_CM_11024] [SWS_CM_11042] [SWS_CM_11043] [SWS_CM_11044] [SWS_CM_11046] [SWS_CM_11047] [SWS_CM_11048] [SWS_CM_11049] [SWS_CM_11050] [SWS_CM_11138] [SWS_CM_11139] [SWS_CM_11262] [SWS_CM_11263] [SWS_CM_11411] [SWS_CM_11412] [SWS_CM_11413] [SWS_CM_11534] [SWS_CM_11535] [SWS_CM_11536] [SWS_CM_11537] [SWS_CM_11538] [SWS_CM_11539] [SWS_CM_11540] [SWS_CM_11541] [SWS_CM_11542] [SWS_CM_11543] [SWS_CM_11544] [SWS_CM_11545] [SWS_CM_11546] [SWS_CM_11547] [SWS_CM_11603] [SWS_CM_11610] [SWS_CM_12004] [SWS_CM_12005] [SWS_CM_80102] [SWS_CM_80103] [SWS_CM_90420]
[RS_CM_00203]	Communication Management shall trigger the application on reception of an event	[SWS_CM_00181] [SWS_CM_00182] [SWS_CM_00183] [SWS_CM_00250] [SWS_CM_00306] [SWS_CM_00309] [SWS_CM_00351] [SWS_CM_00709] [SWS_CM_00710] [SWS_CM_00711] [SWS_CM_10296] [SWS_CM_10328] [SWS_CM_10379] [SWS_CM_10380] [SWS_CM_10515] [SWS_CM_10516] [SWS_CM_10523] [SWS_CM_10534] [SWS_CM_10535] [SWS_CM_11025] [SWS_CM_11026] [SWS_CM_11140] [SWS_CM_11141] [SWS_CM_11534] [SWS_CM_11535] [SWS_CM_11536] [SWS_CM_11537] [SWS_CM_11538] [SWS_CM_11539] [SWS_CM_11540] [SWS_CM_11541] [SWS_CM_11542] [SWS_CM_11543] [SWS_CM_11544] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_11545] [SWS_CM_11546] [SWS_CM_11547] [SWS_CM_11605] [SWS_CM_11606] [SWS_CM_11614] [SWS_CM_11615] [SWS_CM_12007] [SWS_CM_80030] [SWS_CM_80072] [SWS_CM_90420]
[RS_CM_00204]	The Communication Management shall map the protocol independent Service Oriented Communication to the configured protocol binding and shall execute the protocol accordingly.	[SWS_CM_00051] [SWS_CM_00057] [SWS_CM_00058] [SWS_CM_00059] [SWS_CM_00060] [SWS_CM_00061] [SWS_CM_00062] [SWS_CM_00063] [SWS_CM_00064] [SWS_CM_00065] [SWS_CM_00066] [SWS_CM_00067] [SWS_CM_00068] [SWS_CM_00069] [SWS_CM_00070] [SWS_CM_00071] [SWS_CM_00072] [SWS_CM_00073] [SWS_CM_00076] [SWS_CM_00077] [SWS_CM_00078] [SWS_CM_00079] [SWS_CM_00081] [SWS_CM_00082] [SWS_CM_00201] [SWS_CM_00202] [SWS_CM_00203] [SWS_CM_00204] [SWS_CM_00205] [SWS_CM_00206] [SWS_CM_00207] [SWS_CM_00208] [SWS_CM_00209] [SWS_CM_10000] [SWS_CM_10059] [SWS_CM_10060] [SWS_CM_10172] [SWS_CM_10174] [SWS_CM_10240] [SWS_CM_10285] [SWS_CM_10287] [SWS_CM_10288] [SWS_CM_10289] [SWS_CM_10290] [SWS_CM_10291] [SWS_CM_10292] [SWS_CM_10293] [SWS_CM_10294] [SWS_CM_10295] [SWS_CM_10296] [SWS_CM_10297] [SWS_CM_10298] [SWS_CM_10299] [SWS_CM_10300] [SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10303] [SWS_CM_10304] [SWS_CM_10306] [SWS_CM_10307] [SWS_CM_10308] [SWS_CM_10309] [SWS_CM_10310] [SWS_CM_10311] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10314] [SWS_CM_10315] [SWS_CM_10316] [SWS_CM_10317] [SWS_CM_10318] [SWS_CM_10319] [SWS_CM_10320] [SWS_CM_10321] [SWS_CM_10322] [SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10325] [SWS_CM_10326] [SWS_CM_10327] [SWS_CM_10328] [SWS_CM_10330] [SWS_CM_10331] [SWS_CM_10332] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10336] [SWS_CM_10338] [SWS_CM_10339] [SWS_CM_10340] [SWS_CM_10341] [SWS_CM_10342] [SWS_CM_10343] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10346] [SWS_CM_10347] [SWS_CM_10348] [SWS_CM_10349] [SWS_CM_10350] [SWS_CM_10357] [SWS_CM_10358] [SWS_CM_10363] [SWS_CM_10377] [SWS_CM_10378] [SWS_CM_10379] [SWS_CM_10380] [SWS_CM_10381] [SWS_CM_10387] [SWS_CM_10388] [SWS_CM_10389] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_10390] [SWS_CM_10429] [SWS_CM_10430] [SWS_CM_10441] [SWS_CM_10442] [SWS_CM_10444] [SWS_CM_10459] [SWS_CM_10511] [SWS_CM_10512] [SWS_CM_10513] [SWS_CM_10514] [SWS_CM_10515] [SWS_CM_10516] [SWS_CM_10517] [SWS_CM_11000] [SWS_CM_11001] [SWS_CM_11002] [SWS_CM_11040] [SWS_CM_11041] [SWS_CM_11042] [SWS_CM_11043] [SWS_CM_11044] [SWS_CM_11046] [SWS_CM_11047] [SWS_CM_11048] [SWS_CM_11049] [SWS_CM_11050] [SWS_CM_11103] [SWS_CM_11104] [SWS_CM_11105] [SWS_CM_11106] [SWS_CM_11107] [SWS_CM_11108] [SWS_CM_11109] [SWS_CM_11110] [SWS_CM_11111] [SWS_CM_11112] [SWS_CM_11147] [SWS_CM_11148] [SWS_CM_11149] [SWS_CM_11150] [SWS_CM_11151] [SWS_CM_11152] [SWS_CM_11153] [SWS_CM_11154] [SWS_CM_11155] [SWS_CM_11156] [SWS_CM_11269] [SWS_CM_80001] [SWS_CM_80022] [SWS_CM_80064] [SWS_CM_90443] [SWS_CM_90444] [SWS_CM_90445] [SWS_CM_90446] [SWS_CM_90451] [SWS_CM_90452] [SWS_CM_90510] [SWS_CM_90511] [SWS_CM_90512] [SWS_CM_90513] [SWS_CM_90514] [SWS_CM_90515]
[RS_CM_00205]	The Communication Management shall realize the SOME/IP service discovery protocol, the SOME/IP protocol and the E2E supervision (E2E protocol).	[SWS_CM_00090] [SWS_CM_01046] [SWS_CM_10000] [SWS_CM_80001]
[RS_CM_00211]	Communication Management shall provide an interface to provide methods to other applications	[SWS_CM_00052] [SWS_CM_00089] [SWS_CM_00191] [SWS_CM_00199] [SWS_CM_00252] [SWS_CM_00253] [SWS_CM_00254] [SWS_CM_00255] [SWS_CM_00256] [SWS_CM_00257] [SWS_CM_00258] [SWS_CM_00259] [SWS_CM_00260] [SWS_CM_00264] [SWS_CM_00265] [SWS_CM_00301] [SWS_CM_00352] [SWS_CM_10036] [SWS_CM_10037] [SWS_CM_10042] [SWS_CM_10053] [SWS_CM_10054] [SWS_CM_10055] [SWS_CM_10056] [SWS_CM_10057] [SWS_CM_10058] [SWS_CM_10059] [SWS_CM_10060] [SWS_CM_10070] [SWS_CM_10072] [SWS_CM_10076] [SWS_CM_10088] [SWS_CM_10098] [SWS_CM_10099] [SWS_CM_10218] [SWS_CM_10219] [SWS_CM_10222] [SWS_CM_10227] [SWS_CM_10230] [SWS_CM_10234] [SWS_CM_10235] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10248] [SWS_CM_10251] [SWS_CM_10252] [SWS_CM_10253] [SWS_CM_10254] [SWS_CM_10255] [SWS_CM_10256] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_10257] [SWS_CM_10258] [SWS_CM_10259] [SWS_CM_10260] [SWS_CM_10261] [SWS_CM_10262] [SWS_CM_10263] [SWS_CM_10264] [SWS_CM_10265] [SWS_CM_10266] [SWS_CM_10267] [SWS_CM_10268] [SWS_CM_10269] [SWS_CM_10270] [SWS_CM_10271] [SWS_CM_10272] [SWS_CM_10273] [SWS_CM_10274] [SWS_CM_10275] [SWS_CM_10276] [SWS_CM_10277] [SWS_CM_10278] [SWS_CM_10279] [SWS_CM_10280] [SWS_CM_10281] [SWS_CM_10282] [SWS_CM_10283] [SWS_CM_10284] [SWS_CM_10285] [SWS_CM_10361] [SWS_CM_10371] [SWS_CM_10411] [SWS_CM_10459] [SWS_CM_11042] [SWS_CM_11043] [SWS_CM_11044] [SWS_CM_11046] [SWS_CM_11047] [SWS_CM_11048] [SWS_CM_11049] [SWS_CM_11050] [SWS_CM_11262] [SWS_CM_11263] [SWS_CM_11265] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11360] [SWS_CM_11362] [SWS_CM_11550] [SWS_CM_11608] [SWS_CM_11611] [SWS_CM_90501] [SWS_CM_99556]
[RS_CM_00212]	Communication Management shall provide an interface to call methods of other applications synchronously	[SWS_CM_00006] [SWS_CM_00089] [SWS_CM_00192] [SWS_CM_00194] [SWS_CM_00195] [SWS_CM_00196] [SWS_CM_10297] [SWS_CM_10298] [SWS_CM_10299] [SWS_CM_10300] [SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10303] [SWS_CM_10304] [SWS_CM_10306] [SWS_CM_10307] [SWS_CM_10308] [SWS_CM_10309] [SWS_CM_10310] [SWS_CM_10311] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10314] [SWS_CM_10315] [SWS_CM_10316] [SWS_CM_10317] [SWS_CM_10318] [SWS_CM_10329] [SWS_CM_10330] [SWS_CM_10331] [SWS_CM_10332] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10336] [SWS_CM_10338] [SWS_CM_10339] [SWS_CM_10340] [SWS_CM_10341] [SWS_CM_10342] [SWS_CM_10343] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10346] [SWS_CM_10347] [SWS_CM_10348] [SWS_CM_10349] [SWS_CM_10350] [SWS_CM_10371] [SWS_CM_10414] [SWS_CM_10441] [SWS_CM_10442] [SWS_CM_10443] [SWS_CM_10444] [SWS_CM_10447] [SWS_CM_11100] [SWS_CM_11101] [SWS_CM_11102] [SWS_CM_11103] [SWS_CM_11104] [SWS_CM_11105] [SWS_CM_11106] [SWS_CM_11107] [SWS_CM_11108] [SWS_CM_11109] [SWS_CM_11110] [SWS_CM_11111] [SWS_CM_11112] [SWS_CM_11144] [SWS_CM_11145] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_CM_11146] [SWS_CM_11147] [SWS_CM_11148] [SWS_CM_11149] [SWS_CM_11150] [SWS_CM_11151] [SWS_CM_11152] [SWS_CM_11153] [SWS_CM_11154] [SWS_CM_11155] [SWS_CM_11156] [SWS_CM_99332] [SWS_CM_99333] [SWS_CM_99447]
[RS_CM_00213]	Communication Management shall provide an interface to call service methods asynchronously	[SWS_CM_00006] [SWS_CM_00048] [SWS_CM_00049] [SWS_CM_00089] [SWS_CM_00193] [SWS_CM_00194] [SWS_CM_00196] [SWS_CM_00197] [SWS_CM_10297] [SWS_CM_10298] [SWS_CM_10299] [SWS_CM_10300] [SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10303] [SWS_CM_10304] [SWS_CM_10306] [SWS_CM_10307] [SWS_CM_10308] [SWS_CM_10309] [SWS_CM_10310] [SWS_CM_10311] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10314] [SWS_CM_10315] [SWS_CM_10316] [SWS_CM_10317] [SWS_CM_10318] [SWS_CM_10329] [SWS_CM_10330] [SWS_CM_10331] [SWS_CM_10332] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10336] [SWS_CM_10338] [SWS_CM_10339] [SWS_CM_10340] [SWS_CM_10341] [SWS_CM_10342] [SWS_CM_10343] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10346] [SWS_CM_10347] [SWS_CM_10348] [SWS_CM_10349] [SWS_CM_10350] [SWS_CM_10371] [SWS_CM_10414] [SWS_CM_10440] [SWS_CM_10441] [SWS_CM_10442] [SWS_CM_10443] [SWS_CM_10444] [SWS_CM_10447] [SWS_CM_11100] [SWS_CM_11101] [SWS_CM_11102] [SWS_CM_11103] [SWS_CM_11104] [SWS_CM_11105] [SWS_CM_11106] [SWS_CM_11107] [SWS_CM_11108] [SWS_CM_11109] [SWS_CM_11110] [SWS_CM_11111] [SWS_CM_11112] [SWS_CM_11144] [SWS_CM_11145] [SWS_CM_11146] [SWS_CM_11147] [SWS_CM_11148] [SWS_CM_11149] [SWS_CM_11150] [SWS_CM_11151] [SWS_CM_11152] [SWS_CM_11153] [SWS_CM_11154] [SWS_CM_11155] [SWS_CM_11156] [SWS_CM_99332] [SWS_CM_99333] [SWS_CM_99447]
[RS_CM_00214]	Communication Management shall provide an interface to query the result of an asynchronously called service method	[SWS_CM_00048] [SWS_CM_00049] [SWS_CM_00089] [SWS_CM_00193] [SWS_CM_10371] [SWS_CM_10440]
[RS_CM_00215]	Communication Management shall trigger the application on completion of an asynchronously called service method	[SWS_CM_00089] [SWS_CM_00197] [SWS_CM_10317] [SWS_CM_10318] [SWS_CM_10349] [SWS_CM_10350] [SWS_CM_11104] [SWS_CM_11108] [SWS_CM_11148] [SWS_CM_90434]

▽

△

Requirement	Description	Satisfied by
[RS_CM_00216]	Communication Management shall provide an interface which aggregates methods to receive a notification on a changed field value as well as explicitly getting and setting the field value	[SWS_CM_00008] [SWS_CM_00089] [SWS_CM_11403] [SWS_CM_90501] [SWS_CM_99446] [SWS_CM_99558]
[RS_CM_00217]	Communication Management shall provide a method to remotely set the field value	[SWS_CM_00089] [SWS_CM_00113] [SWS_CM_10329] [SWS_CM_10333] [SWS_CM_10335] [SWS_CM_10344] [SWS_CM_10346] [SWS_CM_10417] [SWS_CM_10443] [SWS_CM_11151] [SWS_CM_11152]
[RS_CM_00218]	Communication Management shall provide a method to remotely get the field value	[SWS_CM_00089] [SWS_CM_00112] [SWS_CM_00114] [SWS_CM_00116] [SWS_CM_00119] [SWS_CM_00120] [SWS_CM_00128] [SWS_CM_00129] [SWS_CM_10329] [SWS_CM_10333] [SWS_CM_10335] [SWS_CM_10344] [SWS_CM_10346] [SWS_CM_10412] [SWS_CM_10413] [SWS_CM_10415] [SWS_CM_10417] [SWS_CM_10443] [SWS_CM_11151] [SWS_CM_11152]
[RS_CM_00219]	Communication Management shall provide an interface which aggregates methods to send a notification on value change and to register a get and set function for the field value	[SWS_CM_00007] [SWS_CM_10417] [SWS_CM_11402]
[RS_CM_00220]	Communication Management shall trigger the set method of the application which provides the field	[SWS_CM_10338] [SWS_CM_10339] [SWS_CM_10340] [SWS_CM_11153] [SWS_CM_11154] [SWS_CM_11155] [SWS_CM_11156]
[RS_CM_00221]	Communication Management shall trigger the get method of the application which provides the field	[SWS_CM_10338] [SWS_CM_10339] [SWS_CM_10340] [SWS_CM_11153] [SWS_CM_11154] [SWS_CM_11155] [SWS_CM_11156]
[RS_CM_00223]	The Communication Management shall protect the transmission of events using E2E protocol. The E2E Protection has to be executed behind the event API.	[SWS_CM_00046] [SWS_CM_10473] [SWS_CM_90406] [SWS_CM_90430] [SWS_CM_90433]
[RS_CM_00224]	The communication management shall provide the E2E information of the received event to the application.	[SWS_CM_00042] [SWS_CM_00093] [SWS_CM_00094] [SWS_CM_00095] [SWS_CM_00096] [SWS_CM_00097] [SWS_CM_00098] [SWS_CM_00099] [SWS_CM_00100] [SWS_CM_00105] [SWS_CM_00106] [SWS_CM_00107] [SWS_CM_00108] [SWS_CM_00109] [SWS_CM_00110] [SWS_CM_00126] [SWS_CM_10475] [SWS_CM_11554] [SWS_CM_11612] [SWS_CM_90407] [SWS_CM_90408] [SWS_CM_90412] [SWS_CM_90413] [SWS_CM_90417]
[RS_CM_00225]	Communication Management shall provide an interface to call fire&forget service methods	[SWS_CM_90435] [SWS_CM_90436]

▽

△

Requirement	Description	Satisfied by
[RS_CM_00315]	The Communication Management shall support a change of the configured protocol binding without requiring a re-compilation of the adaptive application	[SWS_CM_10384] [SWS_CM_10385] [SWS_CM_10386]
[RS_CM_00400]	Communication Management shall protect the transmission of methods using E2E protocol.	[SWS_CM_00033] [SWS_CM_00036] [SWS_CM_00037] [SWS_CM_00039] [SWS_CM_00040] [SWS_CM_00041] [SWS_CM_10460] [SWS_CM_10462] [SWS_CM_10463] [SWS_CM_10464] [SWS_CM_10465] [SWS_CM_10466] [SWS_CM_10467] [SWS_CM_10468] [SWS_CM_10469] [SWS_CM_10472] [SWS_CM_10473] [SWS_CM_90467] [SWS_CM_90468] [SWS_CM_90469] [SWS_CM_90470] [SWS_CM_90471] [SWS_CM_90472] [SWS_CM_90473] [SWS_CM_90474] [SWS_CM_90475] [SWS_CM_90476] [SWS_CM_90477] [SWS_CM_90479] [SWS_CM_90480] [SWS_CM_90481] [SWS_CM_90482] [SWS_CM_90485] [SWS_CM_90486] [SWS_CM_90487] [SWS_CM_90488] [SWS_CM_90489] [SWS_CM_90490] [SWS_CM_90491] [SWS_CM_90492] [SWS_CM_90493] [SWS_CM_90494] [SWS_CM_90496] [SWS_CM_90497] [SWS_CM_90498]
[RS_CM_00401]	The communication management shall provide the E2E information of the received method call to the application.	[SWS_CM_00034] [SWS_CM_00047] [SWS_CM_10470] [SWS_CM_10471] [SWS_CM_90495] [SWS_CM_90499]
[RS_CM_00402]	Communication Management shall support a decision for applying the method call based on E2E results.	[SWS_CM_00034] [SWS_CM_00047] [SWS_CM_10467] [SWS_CM_10470] [SWS_CM_10471]
[RS_CM_00500]	Service Contract Version for a Service Interface	[SWS_CM_09004] [SWS_CM_11506] [SWS_CM_11507] [SWS_CM_11508] [SWS_CM_11509] [SWS_CM_11510] [SWS_CM_11511] [SWS_CM_11512] [SWS_CM_11513] [SWS_CM_11514] [SWS_CM_11515] [SWS_CM_11516] [SWS_CM_11517] [SWS_CM_11518] [SWS_CM_11519] [SWS_CM_90508] [SWS_CM_99003] [SWS_CM_99029]
[RS_CM_00501]	Service Contract Versioning for all Transport Deployment Protocols	[SWS_CM_09004] [SWS_CM_11009] [SWS_CM_90508] [SWS_CM_99003]
[RS_CM_00700]	The Service Discovery shall evaluate the service version compatibility for service connection	[SWS_CM_99003]
[RS_CM_00701]	Service Versioning Blocklist	[SWS_CM_10202] [SWS_CM_11009]
[RS_CM_00710]	Static Service Connection	[SWS_CM_02201]

▽

△

Requirement	Description	Satisfied by
[RS_CM_00801]	Secure communication shall be transmitted using secure channels	[SWS_CM_00155] [SWS_CM_10048] [SWS_CM_10049] [SWS_CM_10050] [SWS_CM_11270] [SWS_CM_11271] [SWS_CM_11272] [SWS_CM_11273] [SWS_CM_11274] [SWS_CM_11275] [SWS_CM_11276] [SWS_CM_11277] [SWS_CM_11278] [SWS_CM_11279] [SWS_CM_11280] [SWS_CM_11281] [SWS_CM_11282] [SWS_CM_11283] [SWS_CM_11284] [SWS_CM_11285] [SWS_CM_11286] [SWS_CM_11287] [SWS_CM_11288] [SWS_CM_11289] [SWS_CM_11290] [SWS_CM_11344] [SWS_CM_11345] [SWS_CM_11346] [SWS_CM_11372] [SWS_CM_11378] [SWS_CM_19999] [SWS_CM_90101] [SWS_CM_90102] [SWS_CM_90103] [SWS_CM_90104] [SWS_CM_90108] [SWS_CM_90109] [SWS_CM_90110] [SWS_CM_90115] [SWS_CM_90116] [SWS_CM_90117] [SWS_CM_90118] [SWS_CM_90121] [SWS_CM_90201] [SWS_CM_90202] [SWS_CM_90203] [SWS_CM_90204] [SWS_CM_90205] [SWS_CM_90206] [SWS_CM_90207] [SWS_CM_90209]
[RS_CM_00802]	Secure channels shall be configurable	[SWS_CM_00155] [SWS_CM_10048] [SWS_CM_10049] [SWS_CM_10050] [SWS_CM_11274] [SWS_CM_11276] [SWS_CM_11280] [SWS_CM_11281] [SWS_CM_11282] [SWS_CM_11283] [SWS_CM_11284] [SWS_CM_11285] [SWS_CM_11286] [SWS_CM_11287] [SWS_CM_11288] [SWS_CM_11289] [SWS_CM_11290] [SWS_CM_11344] [SWS_CM_11345] [SWS_CM_11378] [SWS_CM_19999]
[RS_CM_00803]	The assignment of communication to specific secure channels shall be configurable	[SWS_CM_00155] [SWS_CM_10048] [SWS_CM_10049] [SWS_CM_10050] [SWS_CM_10495] [SWS_CM_10496] [SWS_CM_10497] [SWS_CM_11270] [SWS_CM_11280] [SWS_CM_11281] [SWS_CM_11282] [SWS_CM_11283] [SWS_CM_11284] [SWS_CM_11285] [SWS_CM_11286] [SWS_CM_11287] [SWS_CM_11288] [SWS_CM_11289] [SWS_CM_11290] [SWS_CM_11344] [SWS_CM_11345] [SWS_CM_11378] [SWS_CM_19999] [SWS_CM_90102] [SWS_CM_90202]
[RS_CM_00804]	Using secure channels shall be transparent on the communication API	[SWS_CM_00155] [SWS_CM_10048] [SWS_CM_10049] [SWS_CM_10050] [SWS_CM_11280] [SWS_CM_11281] [SWS_CM_11282] [SWS_CM_11283] [SWS_CM_11284] [SWS_CM_11285] [SWS_CM_11286] [SWS_CM_11287] [SWS_CM_11288] [SWS_CM_11289] [SWS_CM_11290] [SWS_CM_11344] [SWS_CM_11345] [SWS_CM_11378] [SWS_CM_19999] [SWS_CM_90111] [SWS_CM_90112] [SWS_CM_90113] [SWS_CM_90114] [SWS_CM_90119]

▽

△

Requirement	Description	Satisfied by
[RS_E2E_08534]	E2E protocol shall provide E2E Check status to the application	[SWS_CM_00038] [SWS_CM_00093] [SWS_CM_00094] [SWS_CM_00095] [SWS_CM_00096] [SWS_CM_00097] [SWS_CM_00098] [SWS_CM_00099] [SWS_CM_00100] [SWS_CM_00105] [SWS_CM_00106] [SWS_CM_00107] [SWS_CM_00108] [SWS_CM_00109] [SWS_CM_00110] [SWS_CM_00126] [SWS_CM_10475] [SWS_CM_11554] [SWS_CM_11612] [SWS_CM_12021] [SWS_CM_12022] [SWS_CM_12026] [SWS_CM_90411] [SWS_CM_90413] [SWS_CM_90416] [SWS_CM_90417] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90478] [SWS_CM_90482] [SWS_CM_90483] [SWS_CM_90484]
[RS_E2E_08540]	E2E protocol shall support protected periodic/mixed periodic communication	[SWS_CM_00042] [SWS_CM_00043] [SWS_CM_00044] [SWS_CM_00045] [SWS_CM_00046] [SWS_CM_90401] [SWS_CM_90402] [SWS_CM_90403] [SWS_CM_90404] [SWS_CM_90406] [SWS_CM_90407] [SWS_CM_90408] [SWS_CM_90410] [SWS_CM_90411] [SWS_CM_90412] [SWS_CM_90413] [SWS_CM_90415] [SWS_CM_90416] [SWS_CM_90417] [SWS_CM_90430] [SWS_CM_90433]
[RS_E2E_08541]	E2E protocol shall support protected non-periodic communication	[SWS_CM_00033] [SWS_CM_00036] [SWS_CM_00037] [SWS_CM_00038] [SWS_CM_00039] [SWS_CM_00040] [SWS_CM_00041] [SWS_CM_10460] [SWS_CM_10462] [SWS_CM_10463] [SWS_CM_10464] [SWS_CM_10465] [SWS_CM_10466] [SWS_CM_10467] [SWS_CM_10468] [SWS_CM_10469] [SWS_CM_10472] [SWS_CM_10473] [SWS_CM_90467] [SWS_CM_90468] [SWS_CM_90469] [SWS_CM_90470] [SWS_CM_90471] [SWS_CM_90472] [SWS_CM_90473] [SWS_CM_90474] [SWS_CM_90475] [SWS_CM_90476] [SWS_CM_90477] [SWS_CM_90478] [SWS_CM_90479] [SWS_CM_90480] [SWS_CM_90481] [SWS_CM_90482] [SWS_CM_90485] [SWS_CM_90486] [SWS_CM_90487] [SWS_CM_90488] [SWS_CM_90489] [SWS_CM_90490] [SWS_CM_90491] [SWS_CM_90492] [SWS_CM_90493] [SWS_CM_90494] [SWS_CM_90495] [SWS_CM_90496] [SWS_CM_90497] [SWS_CM_90498] [SWS_CM_90499]
[RS_IAM_00001]	Limit `Adaptive Application` access to the `Adaptive Platform Foundation` and Services.	[SWS_CM_10498] [SWS_CM_10501] [SWS_CM_10505] [SWS_CM_10506] [SWS_CM_10507] [SWS_CM_10541] [SWS_CM_10542] [SWS_CM_10543] [SWS_CM_90218]

▽

△

Requirement	Description	Satisfied by
[RS_IAM_00002]	Position of Policy Enforcement	[SWS_CM_10493] [SWS_CM_10494] [SWS_CM_10498] [SWS_CM_10501] [SWS_CM_10505] [SWS_CM_10506] [SWS_CM_10507] [SWS_CM_10541] [SWS_CM_10542] [SWS_CM_10543] [SWS_CM_90218]
[RS_IAM_00006]	Access control policies shall be available to the PDP	[SWS_CM_10539] [SWS_CM_90001] [SWS_CM_90003] [SWS_CM_90006]
[RS_IAM_00007]	The `Adaptive Platform Foundation` shall provide access control decisions	[SWS_CM_10539] [SWS_CM_90001] [SWS_CM_90003] [SWS_CM_90006]
[RS_IAM_00010]	Adaptive applications shall only be able to use AUTOSAR Resources when authorized	[SWS_CM_10539] [SWS_CM_90001] [SWS_CM_90003] [SWS_CM_90006]
[RS_SOMEIPSD_-00002]	SOME/IP Service Discovery Protocol shall support unicast messages	[SWS_CM_00206]
[RS_SOMEIPSD_-00003]	SOME/IP Service Discovery Protocol shall support multicast messages	[SWS_CM_00206]
[RS_SOMEIPSD_-00005]	SOME/IP Service Discovery Protocol shall support different versions of the same service	[SWS_CM_00202] [SWS_CM_00203] [SWS_CM_00204] [SWS_CM_00205] [SWS_CM_00206] [SWS_CM_00207] [SWS_CM_00208] [SWS_CM_10378]
[RS_SOMEIPSD_-00006]	SOME/IP Service Discovery Protocol shall define the format of the Service Discovery message	[SWS_CM_00202] [SWS_CM_00203] [SWS_CM_00204] [SWS_CM_00205] [SWS_CM_00206] [SWS_CM_00207] [SWS_CM_00208] [SWS_CM_10377] [SWS_CM_10378] [SWS_CM_10381]
[RS_SOMEIPSD_-00008]	SOME/IP Service Discovery Protocol shall support to find the location of service instances	[SWS_CM_00202] [SWS_CM_00209]
[RS_SOMEIPSD_-00010]	SOME/IP Service Discovery Protocol shall provide support to transport optional data	[SWS_CM_00202] [SWS_CM_00203] [SWS_CM_00204]
[RS_SOMEIPSD_-00013]	SOME/IP Service Discovery Protocol shall support to offer published services	[SWS_CM_00201] [SWS_CM_00203]
[RS_SOMEIPSD_-00014]	SOME/IP Service Discovery Protocol shall support to stop offering services	[SWS_CM_00204]
[RS_SOMEIPSD_-00015]	SOME/IP Service Discovery Protocol shall support to subscribe to events	[SWS_CM_00051] [SWS_CM_00205] [SWS_CM_00206] [SWS_CM_10377] [SWS_CM_10381]
[RS_SOMEIPSD_-00016]	SOME/IP Service Discovery Protocol shall support to deny subscriptions	[SWS_CM_00208]
[RS_SOMEIPSD_-00017]	SOME/IP Service Discovery Protocol shall support to stop subscriptions to events	[SWS_CM_00207] [SWS_CM_10378]
[RS_SOMEIPSD_-00018]	SOME/IP Service Discovery Protocol shall support reboot detection of service providers	[SWS_CM_00050] [SWS_CM_00051]
[RS_SOMEIPSD_-00024]	SOME/IP Service Discovery shall support configurable timings	[SWS_CM_00201] [SWS_CM_00209]
[RS_SOMEIPSD_-00025]	SOME/IP Service Discovery messages shall contain information how to contact the communication partner	[SWS_CM_00203]

▽

△

Requirement	Description	Satisfied by
[RS_SOMEIP_00003]	SOME/IP protocol shall provide support of multiple versions of a service interface	[SWS_CM_00079] [SWS_CM_10291] [SWS_CM_10292] [SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10512] [SWS_CM_10513] [SWS_CM_10519] [SWS_CM_10520] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069]
[RS_SOMEIP_00004]	SOME/IP protocol shall support event communication	[SWS_CM_00076] [SWS_CM_00077] [SWS_CM_00078] [SWS_CM_00079] [SWS_CM_10034] [SWS_CM_10287] [SWS_CM_10288] [SWS_CM_10289] [SWS_CM_10290] [SWS_CM_10291] [SWS_CM_10292] [SWS_CM_10293] [SWS_CM_10294] [SWS_CM_10295] [SWS_CM_10296] [SWS_CM_10319] [SWS_CM_10320] [SWS_CM_10321] [SWS_CM_10322] [SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10325] [SWS_CM_10326] [SWS_CM_10327] [SWS_CM_10328] [SWS_CM_10363] [SWS_CM_10379] [SWS_CM_10380] [SWS_CM_10511] [SWS_CM_10512] [SWS_CM_10513] [SWS_CM_10514] [SWS_CM_10515] [SWS_CM_10516] [SWS_CM_10517] [SWS_CM_10518] [SWS_CM_10519] [SWS_CM_10520] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_10523] [SWS_CM_80021] [SWS_CM_80022] [SWS_CM_80023] [SWS_CM_80024] [SWS_CM_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80030] [SWS_CM_80032] [SWS_CM_80063] [SWS_CM_80064] [SWS_CM_80065] [SWS_CM_80066] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069] [SWS_CM_80072] [SWS_CM_80074]
[RS_SOMEIP_00005]	SOME/IP protocol shall support different strategies for event communication	[SWS_CM_00076] [SWS_CM_10034] [SWS_CM_10287] [SWS_CM_10319] [SWS_CM_10363] [SWS_CM_10511] [SWS_CM_10517] [SWS_CM_10518] [SWS_CM_80021] [SWS_CM_80063]
[RS_SOMEIP_00006]	SOME/IP protocol shall support uni-directional RPC communication	[SWS_CM_10297] [SWS_CM_10298] [SWS_CM_10300] [SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10303] [SWS_CM_10304] [SWS_CM_10306] [SWS_CM_10307] [SWS_CM_10314] [SWS_CM_10441]

▽

△

Requirement	Description	Satisfied by
[RS_SOMEIP_00007]	SOME/IP protocol shall support bi-directional RPC communication	[SWS_CM_10297] [SWS_CM_10298] [SWS_CM_10300] [SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10303] [SWS_CM_10304] [SWS_CM_10306] [SWS_CM_10307] [SWS_CM_10308] [SWS_CM_10309] [SWS_CM_10310] [SWS_CM_10311] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10314] [SWS_CM_10316] [SWS_CM_10317] [SWS_CM_10318] [SWS_CM_10329] [SWS_CM_10330] [SWS_CM_10331] [SWS_CM_10332] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10336] [SWS_CM_10338] [SWS_CM_10339] [SWS_CM_10340] [SWS_CM_10341] [SWS_CM_10342] [SWS_CM_10343] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10346] [SWS_CM_10348] [SWS_CM_10349] [SWS_CM_10350] [SWS_CM_10441] [SWS_CM_10442] [SWS_CM_10443] [SWS_CM_10444] [SWS_CM_10447]
[RS_SOMEIP_00008]	SOME/IP protocol shall support error handling of RPC communication	[SWS_CM_00079] [SWS_CM_10292] [SWS_CM_10302] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10317] [SWS_CM_10334] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10357] [SWS_CM_10358] [SWS_CM_10429] [SWS_CM_10430] [SWS_CM_10513] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_80027] [SWS_CM_80028]
[RS_SOMEIP_00009]	SOME/IP protocol shall support field communication	[SWS_CM_10319] [SWS_CM_10320] [SWS_CM_10321] [SWS_CM_10322] [SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10325] [SWS_CM_10326] [SWS_CM_10327] [SWS_CM_10328] [SWS_CM_10329] [SWS_CM_10330] [SWS_CM_10331] [SWS_CM_10332] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10336] [SWS_CM_10338] [SWS_CM_10339] [SWS_CM_10340] [SWS_CM_10341] [SWS_CM_10342] [SWS_CM_10343] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10346] [SWS_CM_10348] [SWS_CM_10349] [SWS_CM_10350] [SWS_CM_10380] [SWS_CM_10443] [SWS_CM_10444] [SWS_CM_80063] [SWS_CM_80064] [SWS_CM_80065] [SWS_CM_80066] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069] [SWS_CM_80072] [SWS_CM_80074]
[RS_SOMEIP_00010]	SOME/IP protocol shall support different transport protocols underneath	[SWS_CM_00077] [SWS_CM_10288] [SWS_CM_10298] [SWS_CM_10299] [SWS_CM_10309] [SWS_CM_10310] [SWS_CM_10320] [SWS_CM_10330] [SWS_CM_10331] [SWS_CM_10341] [SWS_CM_10342] [SWS_CM_12023] [SWS_CM_12024] [SWS_CM_12025] [SWS_CM_80022] [SWS_CM_80064]

▽

△

Requirement	Description	Satisfied by
[RS_SOMEIP_00012]	SOME/IP protocol shall support session handling	[SWS_CM_10240] [SWS_CM_10301] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10333] [SWS_CM_10344] [SWS_CM_10345]
[RS_SOMEIP_00014]	SOME/IP protocol shall support handling of protocol errors on receiver side	[SWS_CM_00079] [SWS_CM_10292] [SWS_CM_10302] [SWS_CM_10313] [SWS_CM_10324] [SWS_CM_10334] [SWS_CM_10345] [SWS_CM_10428] [SWS_CM_10513] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80069]
[RS_SOMEIP_00017]	SOME/IP protocol shall support grouping events into eventgroups	[SWS_CM_00076] [SWS_CM_10287] [SWS_CM_10319] [SWS_CM_10511] [SWS_CM_10518] [SWS_CM_80021] [SWS_CM_80063]
[RS_SOMEIP_00018]	SOME/IP protocol shall support grouping fields in eventgroups	[SWS_CM_10319] [SWS_CM_80063]
[RS_SOMEIP_00019]	SOME/IP protocol shall identify services using unique identifiers	[SWS_CM_00079] [SWS_CM_10292] [SWS_CM_10302] [SWS_CM_10313] [SWS_CM_10324] [SWS_CM_10334] [SWS_CM_10345] [SWS_CM_10513] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80069]
[RS_SOMEIP_00021]	SOME/IP protocol shall identify RPC methods of services using unique identifiers	[SWS_CM_10301] [SWS_CM_10302] [SWS_CM_10303] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10314] [SWS_CM_10333] [SWS_CM_10334] [SWS_CM_10335] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10346]
[RS_SOMEIP_00022]	SOME/IP protocol shall identify events of services using unique identifiers	[SWS_CM_00079] [SWS_CM_10291] [SWS_CM_10292] [SWS_CM_10293] [SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10325] [SWS_CM_10512] [SWS_CM_10513] [SWS_CM_10514] [SWS_CM_10519] [SWS_CM_10520] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_80025] [SWS_CM_80026] [SWS_CM_80027] [SWS_CM_80028] [SWS_CM_80067] [SWS_CM_80068] [SWS_CM_80069]
[RS_SOMEIP_00025]	SOME/IP protocol shall support the identification of callers of an RPC using unique identifiers	[SWS_CM_10301] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10333] [SWS_CM_10344] [SWS_CM_10345]
[RS_SOMEIP_00026]	SOME/IP protocol shall define the endianness of header and payload	[SWS_CM_10172] [SWS_CM_80003]
[RS_SOMEIP_00028]	SOME/IP protocol shall specify the serialization algorithm for data	[SWS_CM_10034] [SWS_CM_10294] [SWS_CM_10304] [SWS_CM_10316] [SWS_CM_10326] [SWS_CM_10336] [SWS_CM_10348] [SWS_CM_10442] [SWS_CM_10444] [SWS_CM_80032] [SWS_CM_80074]
[RS_SOMEIP_00037]	SOME/IP protocol shall support transporting array data types with flexible length	[SWS_CM_00270]

▽

△

Requirement	Description	Satisfied by
[RS_SOMEIP_00041]	SOME/IP protocol shall provide support of multiple versions of the protocol	[SWS_CM_10291] [SWS_CM_10301] [SWS_CM_10312] [SWS_CM_10313] [SWS_CM_10323] [SWS_CM_10333] [SWS_CM_10344] [SWS_CM_10345] [SWS_CM_10512] [SWS_CM_10519] [SWS_CM_10520] [SWS_CM_80025] [SWS_CM_80026] [SWS_CM_80067] [SWS_CM_80068]
[RS_SOMEIP_00042]	SOME/IP protocol shall support unicast and multicast based event communication	[SWS_CM_00078] [SWS_CM_10289] [SWS_CM_10290] [SWS_CM_10321] [SWS_CM_10322] [SWS_CM_80023] [SWS_CM_80024] [SWS_CM_80065] [SWS_CM_80066]
[RS_SOMEIP_00050]	SOME/IP protocol shall support serialization of extensible data structs	[SWS_CM_01046]
[RS_SOMEIP_00051]	SOME/IP protocol shall provide support for segmented transmission of large data	[SWS_CM_10445] [SWS_CM_10454] [SWS_CM_10455] [SWS_CM_10456] [SWS_CM_10457] [SWS_CM_99036] [SWS_CM_99037] [SWS_CM_99038] [SWS_CM_99039]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 General description

The AUTOSAR Adaptive architecture organizes the software of the AUTOSAR Adaptive foundation as functional clusters. These clusters offer common functionality as services to the applications. The Communication Management (CM) for AUTOSAR Adaptive is such a functional cluster and is part of "AUTOSAR Runtime for Adaptive Applications" - ARA. It is responsible for the construction and supervision of communication paths between applications, both local and remote.

The CM provides the infrastructure that enables communication between Adaptive AUTOSAR Applications within one machine and with software entities on other machines, e.g. other Adaptive AUTOSAR applications or Classic AUTOSAR SWCs. All communication paths can be established at design- , start-up- or run-time.

This specification includes the syntax of the API, the relationship of API to the model and describes semantics, e.g. through state machines, and assumption of pre-, post-conditions and use of APIs. The specification does not provide constraints on the SW architecture of a platform implementation, so there is no definition of basic software modules and no specification of implementation or internal technical architecture of the Communication Management.

7.1.1 Architectural concepts

The Communication management of AUTOSAR Adaptive can be logically divided into the following sub-parts:

- Language binding
- End-to-end communication protection
- Communication / Network binding
- Communication Management software

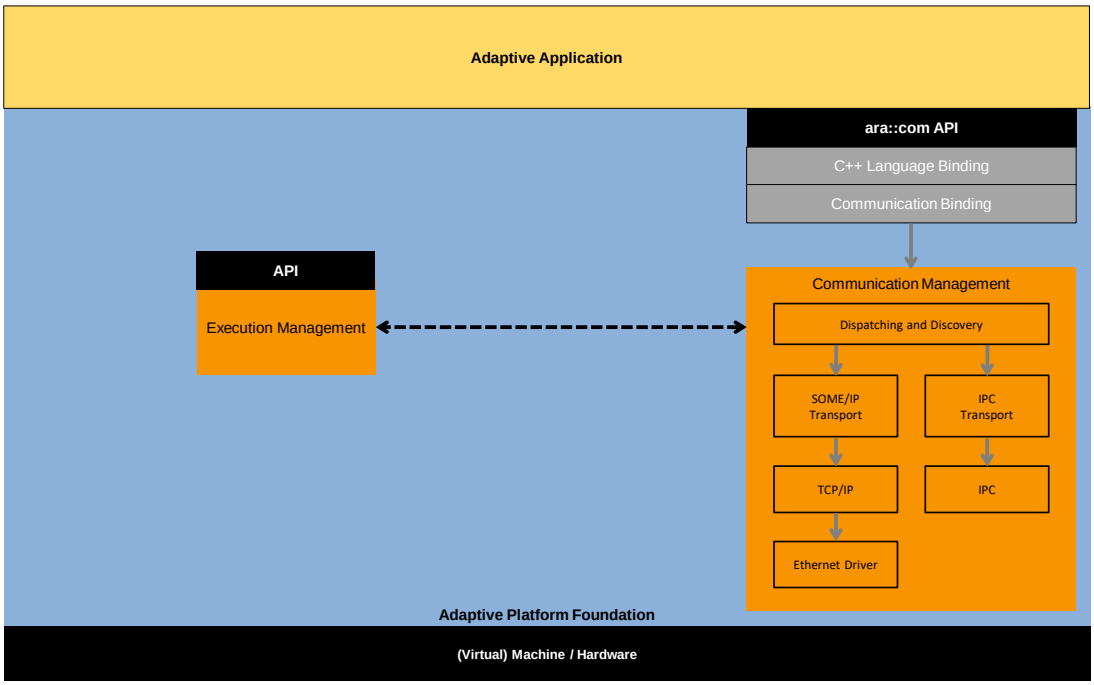

Figure 7.1: Technical Architecture of Communication Management

In the context of Communication Management, the following types of interfaces are defined:

- Public Application Interface: Part of the Adaptive AUTOSAR API and specified in the SWS. This is the standardized ara::com API.
- Functional Cluster Interactions: Interaction between functional clusters. Not normative, intended to make specification more readable and to support integration of SW into demonstrator. (dotted arrow in 7.1) And also interactions between elements within a functional cluster. Not used in specifications, so it is a non-standardized interface. Used for communication inside Communication Management software (grey arrow in 7.1)

Please note, that Language Binding and Communication Binding depend on a specific configuration by the integrator, but they need to be deployed within the application binary. This results in the fact that the serialization of the Communication Binding will run in the execution context of the Adaptive Application.

For the design of ARA API the following constraints apply:

- Support the independence of application software components
- Use of Service-oriented communication without dependency on a specific communication protocol

- Make the API as lean as possible, neither supporting very specific use cases which could also be done on top of the API, nor supporting component model or higher level concepts. The API is restricted to support core communication mechanisms.
- Support for dynamic communication:
  - No discovery by application middleware, the clients know the server but the Server does not know the clients. Event subscription is the only dynamic communication pattern in the application.
  - Full service discovery in the application. No communication paths are known at configuration time. An API for Service discovery allows the application code to choose the service instance.
- Support both Event/Callback and Polling style usage of the API to enable classic RTE style paradigms. To support high determinism demands in case of callback-based / event-based interaction, there shall be the possibility to avoid uncontrolled context switches.
- Support both synchronous callback-based communication and asynchronous communication philosophy.
- Support of client/server communication.
- Support of sender/receiver communication with queued semantics where the receiver caches are configurable.
- Support of selection of trigger conditions for task activation.
- Extensions for security.
- Extensions for Quality Of Service QoS.
- Scalability for real-time systems.
- Support of built-in end-to-end communication protection, where a use-case-specific behavior can be done on top of ARA API.

7.1.2 Design decisions

The design of the ARA API covers the following principles:

- It uses the Proxy/Skeleton pattern:
  - The (service) proxy is the representative of the possibly remote (i.e. other process, other core, other node) service. It is an instance of a C++ class local to the application/client, which uses the service.
  - The (service) skeleton is the connection of the user provided service implementation to the middleware transport infrastructure. Service implementation class is derived from the (service) skeleton.

  - Beside proxies/skeletons, there might exist a so-called "Runtime" (singleton) class to provide some essentials to manage proxies and skeletons. But this is communication management software implementation specific and therefore not specified in this document, but may be specified in a future version.

  Regarding proxy/skeleton design pattern in general and its role in middleware implementations, see [12] [13].

- It supports callback mechanisms on data reception.
- The API has zero-copy capabilities including the possibility for memory management in the middleware.
- It is aligned with the AUTOSAR service model (services, instances, events, methods, ...) to allow the generation of proxies and skeletons out of this model.
- Full discovery and service instance selection support on API level.
- Client/Server Communication uses concepts introduced by C++11 language, e.g. std::future, std::promise, to fully support method calls between different contexts.
- Abstract from SOME/IP specific behavior, but support SOME/IP service mechanisms, as methods, events and fields.
- Support/implement the standard end-to-end protection protocols, as specified in [9] and [7].
- Support of Service contract versioning.
- Support Event and Polling style usage of the API equally to enable classic RT style paradigms.
- Fully exploit C++11/14 features in API design to provide usability and comfort for the application developer.

See ARAComAPI explanatory [1] for more details and explanations on the ARA API design.

7.1.3 Communication paradigms and Definitions

Service-Oriented Communication (SoC) as a part of Service-Oriented Architecture (SOA) [14] is the main communication pattern for Adaptive AUTOSAR Applications. It allows establishing communication paths both at run-time, so it can be used to build up dynamic communication with unknown number of participants. Figure 7.2 shows the basic operation principle of Service-Oriented Communication.

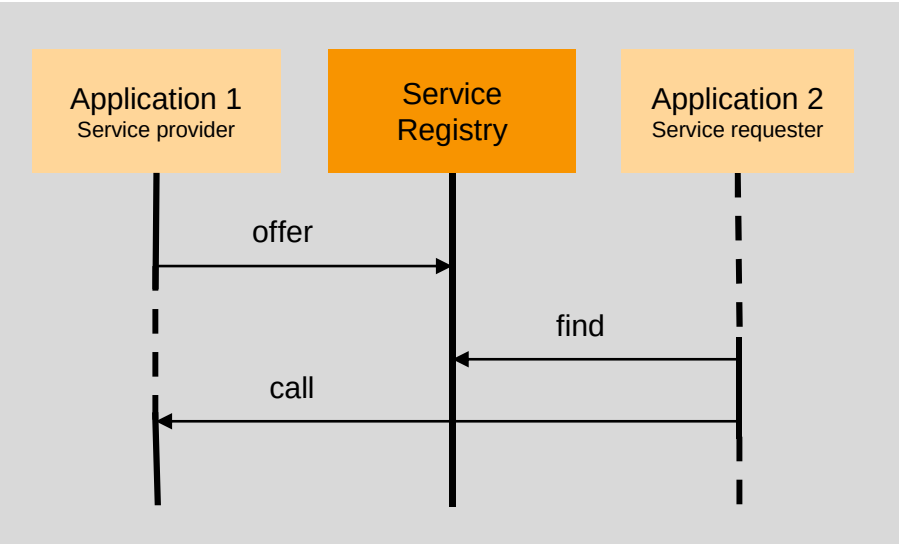

Figure 7.2: Service-Oriented Communication

Service Discovery decides whether external and internal service-oriented communication is established. The discovery strategy shall allow either returning a specific service instance or all available instances providing the requested service at the time of the request, no matter if they are available locally or remote. The Communication Management software should provide an optimized implementation for both the Service discovery and the communication connection, depending on the location where the service provider resides. More about Service Discovery can be found in *SOME/IP Service Discovery Protocol Specification* [6].

The service class is the central element of the Service-Oriented Communication pattern applied in Adaptive AUTOSAR. It represents the service by collecting the methods and events which are provided or requested by the applications implementing the concrete service functionality.

[SWS_CM_12002] Active subscriber

Upstream requirements: RS_CM_00103, RS_AP_00114, RS_AP_00120

⌈The active subscriber shall be an adaptive application that has invoked the Subscribe method of the respective:

- Trigger class (see [SWS_CM_00723]) or
- Field or Event class (see [SWS_CM_00141])

and has not canceled the subscription by invoking the Unsubscribe method of the respective:

- Trigger class (see [SWS_CM_00810]) or
- Field or Event class (see [SWS_CM_00151])

⌋

[SWS_CM_12003] Active subscriber when SOME/IP Network binding is used

Upstream requirements: RS_CM_00103, RS_AP_00114, RS_AP_00120

⌈In addition to [SWS_CM_12002], if SOME/IP Network binding is used to provide services for an application, the active subscriber shall be an adaptive application for which the SOME/IP services subscription has not yet expired when the TTL contained in the respective SOME/IP SubscribeEventgroup message has been exceeded (see [SWS_CM_00205]).⌋

7.1.4 Service contract versioning

In Service Oriented Architecture (SOA) environments the client and the provider of a service rely on a contract which covers the service interface and behavior. The interface and the behavior of a service may change over time. Therefore, service contract versioning has been introduced to differentiate between the different versions of a service.

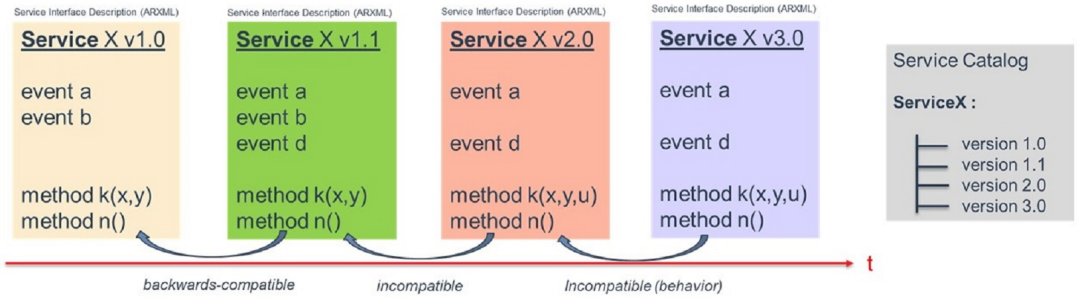

Figure 7.3: Service contract versioning over time

The AUTOSAR Adaptive platform supports service contract versioning. The service contract versioning is separated between the design phase and the deployment phase. This means that any service at design level may have its own version number which is mapped to a version number of the used network binding and vice versa. The mapping process is manually done by the service designer or integrator.

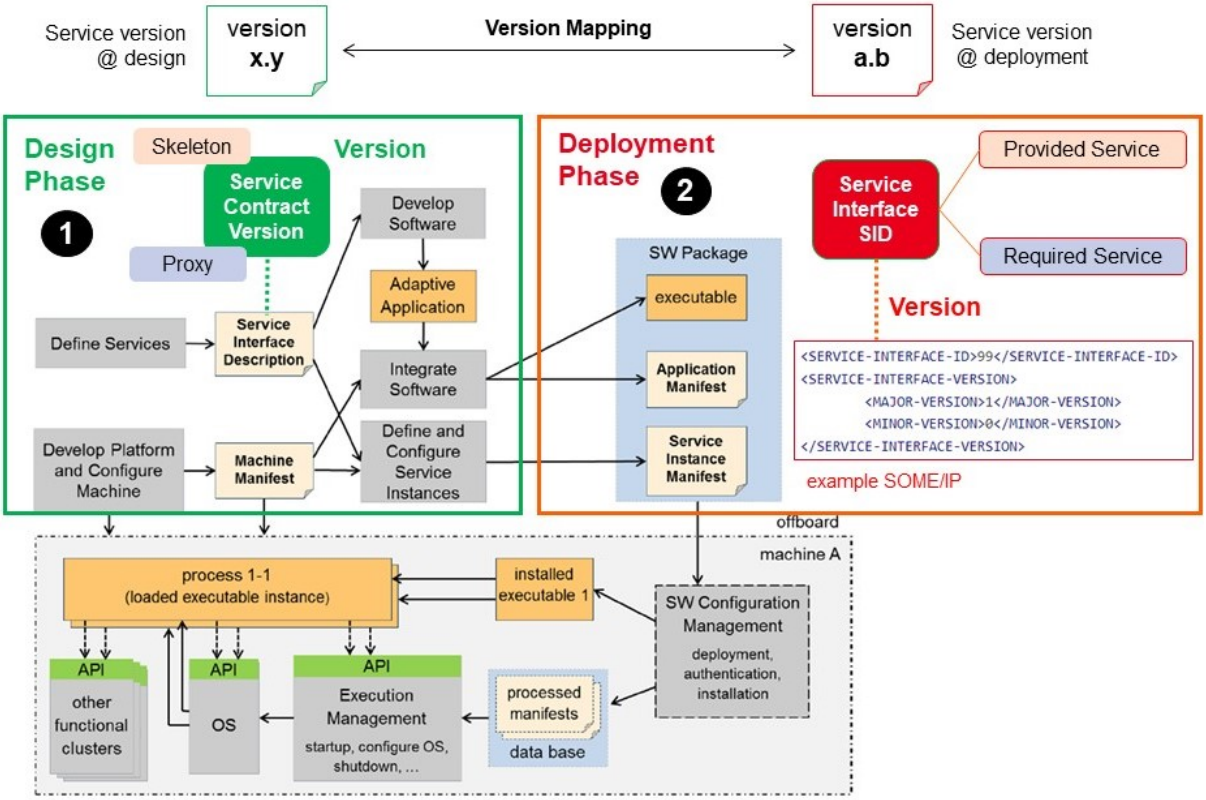

Figure 7.4: Service contract versioning flow

Note:

1. The contract version of a `ServiceInterface` consists of a `majorVersion` and a `minorVersion` number. The `majorVersion` number indicates backwards-incompatible service changes. The `minorVersion` number indicates backwards-compatible service changes.

   - for backwards-incompatible interface or behavior changes the `majorVersion` number is increased and the `minorVersion` number is set to 0.
   - for backwards-compatible interface or behavior changes the `majorVersion` number is unchanged and the `minorVersion` number is increased.

2. The contract version of a `ServiceInterface` is mapped to a version of the `ServiceInterfaceDeployment`. This version mapping may be done several times resulting in several `ServiceInterfaceDeployment`s for the same `ServiceInterface`. Such a mapping will result in unambiguous identification on each VLAN according to the [constr_1723] in [5].

[SWS_CM_99003] Service interface version evaluation for backwards-compatibility

Upstream requirements: RS_CM_00500, RS_CM_00501, RS_CM_00700

⌈The version of `ServiceInterfaceDeployment` shall be evaluated by the Service Discovery in terms of backwards-compatibility based on the used network binding for service connection.⌋

7.2 Network binding

The following chapters describe the requirements according to specific network protocol bindings.

Since the selection of a particular network protocol binding is an integrator driven deployment decision, any change in the selection of a particular network protocol binding or changes in the various attributes and parameters of a particular network protocol binding shall be possible without requiring a re-compilation of the involved adaptive applications. The required changes to the involved adaptive application shall be limited to a re-linking (either static or dynamic) of the involved adaptive application.

[SWS_CM_10384] Change of Service Interface Deployment

Upstream requirements: RS_CM_00315

⌈A change of the service interface deployment shall be possible without re-compiling the involved adaptive applications. – This means that the following changes in the service interface deployment shall be possible without the need for a re-compilation of the adaptive applications:

- changes to the concrete type of `ServiceInterfaceDeployment` and the composed `ServiceMethodDeployment`, `ServiceFieldDeployment`, and `ServiceEventDeployment` (e.g., changing a `SomeipServiceInterfaceDeployment` to a `UserDefinedServiceInterfaceDeployment`)
- changes to one or more attributes of meta-classes derived from `ServiceInterfaceDeployment`, `ServiceMethodDeployment`, `ServiceFieldDeployment`, and `ServiceEventDeployment` (e.g., changing the value of `SomeipEventDeployment.separationTime`)
- backwards-compatible changes to the technology specific service version number of the `ServiceInterfaceDeployment`.

⌋

Note that changes to `SomeipServiceVersion.majorVersion` are an exception here, since any change to `SomeipServiceVersion.majorVersion` indicates an in-

compatible change of the ServiceInterface and thus affects the involved adaptive applications mandating a re-compilation of the involved adaptive applications.

[SWS_CM_10385] Change of Service Instance Deployment

Upstream requirements: RS_CM_00315

⌈A change of the service instance deployment shall be possible without re-compiling the involved adaptive applications. – This means that the following changes in the service instance deployment shall be possible without the need for a re-compilation of the adaptive applications:

- changes to the concrete type of ProvidedApServiceInstance and/or RequiredApServiceInstance (e.g., changing a ProvidedSomeipServiceInstance to a ProvidedUserDefinedServiceInstance and a RequiredSomeipServiceInstance to a RequiredUserDefinedServiceInstance)
- changes to one or more attributes of meta-class derived from ProvidedApServiceInstance and/or RequiredApServiceInstance (e.g., changing the value of the SomeipProvidedEventGroup.multicastThreshold or the SomeipSdServerServiceInstanceConfig.serviceOfferTimeToLive).
- backwards-compatible changes to the technology specific service version number of the ServiceInterfaceDeployment.

⌋

Note that changes to SomeipServiceVersion.majorVersion are an exception here, since any change to SomeipServiceVersion.majorVersion indicates an incompatible change of the ServiceInterface and thus affects the involved adaptive applications mandating a re-compilation of the involved adaptive applications.

[SWS_CM_10386] Change of Network Configuration

Upstream requirements: RS_CM_00315

⌈A change of the network configuration shall be possible without re-compiling the involved adaptive applications. – This means that the following changes in the network configuration shall be possible without the need for a re-compilation of the adaptive applications:

- changes to one or more attributes of a concrete ServiceInstanceToMachineMapping (e.g., changing the value of the SomeipServiceInstanceToMachineMapping.udpPort or the SomeipServiceInstanceToMachineMapping.tcpPort).

⌋

Abstract network protocol bindings for service ports shall be specified inside the service instance manifest to deploy network bindings of service instances.

[SWS_CM_10590] Abstract Network Protocol Binding

Upstream requirements: RS_CM_00200, RS_AP_00137

⌈The usage of abstract network protocol binding for `ProvidedApServiceInstance` and `RequiredApServiceInstance` shall be supported to deploy network bindings of `ServiceInterface`s. An abstract network protocol binding shall cover `SOME/IP`, `DDS` and `UserDefined` protocols and is specified inside the service instance manifest. It is used with an `ara::core::InstanceSpecifier` and shall be specified as followed:
`<port context>::<port name>`, where:

- `<port context>` specifies the instantiation context of the port which might be an instantiation path or any other unique identifiable information.
- `<port name>` specifies the port name.

Note: it is possible to specify multiple technology bindings for a port (Multi-Binding).⌋

[SWS_CM_10416] Reception of a malformed message

Upstream requirements: RS_CM_00005

⌈In case any network binding does receive a message, which it identifies as malformed, the message shall be discarded and the error shall not be propagated to the application.⌋

Note: The incident should also be logged if logging is configured and the corresponding network binding supports it.

7.2.1 SOME/IP Network binding

SOME/IP supports different kind of bindings:

SOME/IP Events:

- unicast is one-to-one communication
- multi-cast is one-to-many communication

In case the active subscriptions will reach the multi-cast-threshold the communication paradigm will be switched from unicast to multi-cast to gain a better network utilization. Below the multi-cast-threshold `SOME/IP` is maintaining for a subscription a single unicast communication.

SOME/IP Events:

- many-to-one communication using multiple unicast communications

[SWS_CM_10000] SOME/IP Compliance

Upstream requirements: RS_CM_00204, RS_CM_00205

⌈The SOME/IP network binding shall implement the SOME/IP Protocol and the SOME/IP Service Discovery Protocol defined in [4] and [6].⌋

The byte order of the SOME/IP header fields is defined as network byte order by [PRS_SOMEIP_00368].

The byte order of additional fields in the SOME/IP payload is defined as network byte order by [PRS_SOMEIP_00759].

[SWS_CM_10172] Payload Byte order definition

Upstream requirements: RS_CM_00204, RS_SOMEIP_00026

⌈The byte order of the parameters inside the payload shall be defined according to [PRS_SOMEIP_00369] by `byteOrder` of `ApSomeipTransformationProps`.⌋

[SWS_CM_10240] Session handling state

Upstream requirements: RS_CM_00204, RS_SOMEIP_00012

⌈In case of normal (i.e., non Fire and Forget) method calls or getters and setters of `Fields` (i.e., in case of SOME/IP messages of type REQUEST, RESPONSE, and ERROR) or if segmentation of SOME/IP messages needs to be performed (i.e. [SWS_CM_10454] and [SWS_CM_10455] and [SWS_CM_10456] apply and [SWS_CM_10457] does not apply) the Session handling shall be Active. Otherwise, the Session handling shall be Inactive.⌋

Note: Segmentation of SOME/IP messages according to section 7.2.1.9.9.

7.2.1.1 Static Service Connection

[SWS_CM_02201] Static service connection

Upstream requirements: RS_CM_00710

⌈The static connection of services which are bound to SOME/IP protocols shall be preformed by statically pre-configured application end-points as described in the TPS_ManifestSpecification for a `ProvidedSomeipServiceInstance` by [TPS_MANI_03312], [TPS_MANI_03313] and for a `RequiredSomeipServiceInstance` by [TPS_MANI_03314], [TPS_MANI_03315], [TPS_MANI_03316].⌋

[SWS_CM_02202] Service Discovery is bypassed by static service connection
⌈The service discovery protocols are bypassed in case of a static service connection.⌋

[SWS_CM_02203] Service versioning is not checked at runtime in case of a static service connection ⌈Service versions are not checked at run-time in case of a static service connection since the Service Discovery has been bypassed.⌋

Note: ara::com language APIs are agnostic to static service connection.

7.2.1.2 Service Discovery

[SWS_CM_00050] Implement reboot detection

Upstream requirements: RS_SOMEIPSD_00018

⌈The service discovery of the SOME/IP network binding shall implement the reboot detection according to [PRS_SOMEIPSD_00254], [PRS_SOMEIPSD_00255], [PRS_SOMEIPSD_00256], [PRS_SOMEIPSD_00631], [PRS_SOMEIPSD_00258], and [PRS_SOMEIPSD_00503].⌋

7.2.1.2.1 Start of service discovery protocol

[SWS_CM_11374] Periodic link state monitoring ⌈The SOME/IP network binding shall periodically monitor and obtain the current link state of the underlying network interfaces.

Note: This information is required since the behavior of SOME/IP service discovery is influenced by the current link state as well as by changes in the link state⌋

[SWS_CM_00201] Start of service discovery protocol on Server side

Upstream requirements: RS_CM_00204, RS_CM_00101, RS_SOMEIPSD_00024, RS_SOMEIPSD_00013

⌈The registration of a new offered service which is bound to SOME/IP by invoking the `OfferService()` method (see [SWS_CM_00101]) of the `ServiceSkeleton` class shall trigger the start of the initial wait phase of the SOME/IP service discovery protocol after link up according to [PRS_SOMEIPSD_00133].⌋

The different phases of SOME/IP Service Discovery on the Server side are configured in the Manifest in the `SomeipSdServerServiceInstanceConfig` referenced in `ProvidedSomeipServiceInstance` element in the role `sdServerConfig`. The configuration is described in more detail in TPS_ManifestSpecification by

- [TPS_MANI_03012] (Initial Wait Phase),
- [TPS_MANI_03013] (Repetition Phase),

- [TPS_MANI_03014] (Main Phase).

The corresponding timing parameters for these phases are configured via `InitialSdDelayConfig` in the role `initialOfferBehavior`, `RequestResponseDelay` in the role `requestResponseDelay`, and `TimeValue` in attribute `offerCyclicDelay`. The sharing of timers is described in [TPS_MANI_03230].

[SWS_CM_00209] Start of service discovery protocol on Client side

Upstream requirements: RS_CM_00204, RS_CM_00102, RS_SOMEIPSD_00024, RS_SOMEIPSD_00008

⌈When invoking the `FindService(InstanceIdentifier)` ([SWS_CM_00122]) / `FindService(InstanceSpecifier)` ([SWS_CM_00622]) methods or the `StartFindService(InstanceIdentifier)` ([SWS_CM_00123]) / `StartFindService(InstanceSpecifier)` ([SWS_CM_00623]) methods of the `ServiceProxy` class, such a search request shall be considered as issuing an internal service request as used in [PRS_SOMEIPSD_00435]. FindService shall not wait for offer messages, but only check information available within the local AP-instance (StartFindService shall also not wait for offer messages as it only registers a handler).⌋

Note: The result of a FindService call depends on the already received offers, hence multiple calls might be necessary to find a service instance at all. Also, the number of found service instances might vary for subsequent calls of FindService.

Note for [SWS_CM_00201] and [SWS_CM_00209]: See also [PRS_SOMEIPSD_00395], [PRS_SOMEIPSD_00397], [PRS_SOMEIPSD_00399], [PRS_SOMEIPSD_00416], [PRS_SOMEIPSD_00435], [PRS_SOMEIPSD_00752], [PRS_SOMEIPSD_00133], [PRS_SOMEIPSD_00805] and [PRS_SOMEIPSD_00751].

The different phases of SOME/IP Service Discovery on the Client side are configured in the Manifest in the `SomeipSdClientServiceInstanceConfig` referenced in `RequiredSomeipServiceInstance` element in the role `sdClientConfig`. The configuration is described in more detail in TPS_ManifestSpecification by

- [TPS_MANI_03026] (Initial Wait Phase),
- [TPS_MANI_03027] (Repetition Phase).

The corresponding timing parameters for these phases are configured via `InitialSdDelayConfig` in the role `initialFindBehavior`, and `RequestResponseDelay` in the role `requestResponseDelay`. The sharing of timers is described in [TPS_MANI_03231].

7.2.1.2.2 FindService message

[SWS_CM_00202] SOME/IP FindService message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00102, RS_SOMEIPSD_00006, RS_SOMEIPSD_00005, RS_SOMEIPSD_00008, RS_SOMEIPSD_00010

⌈The fields in the SOME/IP FindService message shall be as follows:

- The Type field and the TTL field shall be set to values suitable for a FindService entry, which means that
  - The Type field shall be set to FindService (see [PRS_SOMEIPSD_00351] for numerical value)
  - TTL for FindService messages shall not be used, and the value may be set to an arbitrary value. The field is only defined in the protocol for backward compatibility.
- The Service ID field shall be set to a value derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Instance ID shall be set to a value derived from the Manifest where the `RequiredSomeipServiceInstance` element defines the `requiredServiceInstanceId` for the `SomeipServiceInterfaceDeployment` that is referenced by the `RequiredSomeipServiceInstance` in the role `serviceInterfaceDeployment`. If the `requiredServiceInstanceId` is set to "ALL" then 0xFFFF shall be used.
- The Major Version field of the `RequiredSomeipServiceInstance` that is searched shall be set to a value derived from the Manifest where the `SomeipServiceVersion` element that is aggregated by the `SomeipServiceInterfaceDeployment` in the role `serviceInterfaceVersion` defines the `majorVersion`.
- The Minor Version field of the `RequiredSomeipServiceInstance` that is searched shall be set to a value derived from the Manifest from the `requiredMinorVersion` attribute in the `RequiredSomeipServiceInstance`.
  - If `versionDrivenFindBehavior` is set to `minimumMinorVersion` then the Minor Version Field shall be set to 0xFFFF FFFF and all found services with a minor version smaller than the `requiredMinorVersion` shall not be considered for service discovery.
  - If `versionDrivenFindBehavior` is set to `exactOrAnyMinorVersion` then the Minor Version Field shall be set with the `requiredMinorVersion`.
  - If the `minorVersion` is set to "ALL", then the Minor Version Field shall be set to 0xFFFF FFFF.

- Configuration Option shall be used in the find message if at least one capabilityRecord is defined in the RequiredSomeipServiceInstance element. The content of the Configuration Option shall be derived from the key/value pairs defined in each capabilityRecord.

⌋

[SWS_CM_10202] Version blocklist

Upstream requirements: RS_CM_00701

⌈The service connection of a RequiredSomeipServiceInstance with a certain SomeipServiceVersion shall not be considered for service discovery for this instance if this SomeipServiceVersion is listed inside a RequiredSomeipServiceInstance.blocklistedVersion.⌋

7.2.1.2.3 OfferService message

[SWS_CM_00203] SOME/IP OfferService message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00101, RS_SOMEIPSD_00006, RS_SOMEIPSD_00005, RS_SOMEIPSD_00010, RS_SOMEIPSD_00013, RS_SOMEIPSD_00025

⌈The fields in the SOME/IP OfferService message shall be as follows:

- The Type field and the TTL field shall be set to values suitable for a OfferService entry, which means that
  - The Type field shall be set to OfferService (see [PRS_SOMEIPSD_00356] for numerical value).
  - The TTL field shall be set to a value derived from the Manifest where the SomeipSdServerServiceInstanceConfig element that is referenced by the ProvidedSomeipServiceInstance in the role sdServerConfig defines the serviceOfferTimeToLive.
- The Service ID field shall be set to a value derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Instance ID shall be set to a value derived from the Manifest where the ProvidedSomeipServiceInstance element defines the serviceInstanceId for the SomeipServiceInterfaceDeployment that is referenced by the ProvidedSomeipServiceInstance in the role serviceInterfaceDeployment.
- Major Version field of the SomeipServiceInterfaceDeployment that is offered shall be set to a value derived from the Manifest where the SomeipSer-

viceVersion element that is aggregated by the SomeipServiceInterfaceDeployment in the role serviceInterfaceVersion defines the majorVersion.

- Minor Version field of the SomeipServiceInterfaceDeployment that is offered shall be set to a value derived from the Manifest where the SomeipServiceVersion element that is aggregated by the SomeipServiceInterfaceDeployment in the role serviceInterfaceVersion defines the minorVersion.
- The Endpoint Option(s) shall be set in the following way:
  - An IPv4 Endpoint Option shall be used if the Machine to which the ProvidedSomeipServiceInstance is mapped with the ServiceInstanceToMachineMapping provides an EthernetCommunicationConnector that refers to a NetworkEndpoint in the role unicastNetworkEndpoint where an IPv4 Address is configured in the Ipv4Configuration element.
  - An IPv6 Endpoint Option shall be used if the Machine to which the ProvidedSomeipServiceInstance is mapped with the ServiceInstanceToMachineMapping provides an EthernetCommunicationConnector that refers to a NetworkEndpoint in the role unicastNetworkEndpoint where an IPv6 Address is configured in the Ipv6Configuration element.
  - The Transport Layer Protocol used in the IPv4 Endpoint option and/or IPv6 Endpoint option shall be derived from the Manifest where the SomeipServiceInstanceToMachineMapping element that maps the ProvidedSomeipServiceInstance to an EthernetCommunicationConnector of a Machine defines the transport protocol and the port number.
    - UDP shall be used if SomeipServiceInstanceToMachineMapping.udpPort is configured.
    - TCP shall be used ifSomeipServiceInstanceToMachineMapping.tcpPort is configured. In case the port number (SomeipServiceInstanceToMachineMapping.udpPort or SomeipServiceInstanceToMachineMapping.tcpPort) is configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.

⌋

[SWS_CM_12019] Service Discovery Endpoint Options

Upstream requirements: RS_CM_00101

⌈The SOME/IP-SD implementation shall support [PRS_SOMEIPSD_00547], [PRS_SOMEIPSD_00650], [PRS_SOMEIPSD_00651], [PRS_SOMEIPSD_00548], [PRS_-

SOMEIPSD_00549], [PRS_SOMEIPSD_00550], [PRS_SOMEIPSD_00551], [PRS_SOMEIPSD_00552], [PRS_SOMEIPSD_00856], [PRS_SOMEIPSD_00857], [PRS_SOMEIPSD_00854] in case of IPv4.

[PRS_SOMEIPSD_00554], [PRS_SOMEIPSD_00654], [PRS_SOMEIPSD_00555], [PRS_SOMEIPSD_00556], [PRS_SOMEIPSD_00557], [PRS_SOMEIPSD_00558], [PRS_SOMEIPSD_00559], [PRS_SOMEIPSD_00837], [PRS_SOMEIPSD_00859], [PRS_SOMEIPSD_00860], [PRS_SOMEIPSD_00855] in case of IPv6.⌋

Note: The sending of the SD Endpoint Options is currently out of scope of AUTOSAR.

[SWS_CM_11373] Cyclic interval of OfferService messages ⌈If attribute `SomeipSdServerServiceInstanceConfig.offerCyclicDelay` is configured in `SomeipSdServerServiceInstanceConfig` and is greater than 0, in the Main Phase an OfferService entry shall be sent cyclically with an interval defined by configuration item `SomeipSdServerServiceInstanceConfig.offerCyclicDelay`.

If `SomeipSdServerServiceInstanceConfig.offerCyclicDelay` is 0, no OfferService entries shall be sent in Main Phase for this Server Service Instance.⌋

7.2.1.2.4 StopOfferService message

[SWS_CM_00204] SOME/IP StopOffer message

Upstream requirements: RS_CM_00204, RS_CM_00105, RS_SOMEIPSD_00006, RS_SOMEIPSD_00005, RS_SOMEIPSD_00010, RS_SOMEIPSD_00014

⌈The fields in the SOME/IP StopOffer message shall be as follows:

- The Type field and the TTL field shall be set to values suitable for a StopOffer entry, which means that
  - The Type field shall be set to OfferService (see [PRS_SOMEIPSD_00356] for numerical value)
  - The TTL fields shall be set to 0x000000 (see [PRS_SOMEIPSD_00364])
- The Service ID field shall be set to the same value as in the OfferService message.
- The Instance ID field shall be set to the same value as in the OfferService message.
- The Major Version field shall be set to the same value as in the OfferService message.
- The Minor Version field shall be set to the same value as in the OfferService message.

- IPv4 Endpoint Option shall be set to the same value as in the OfferService message.
- IPv6 Endpoint Option shall be set to the same value as in the OfferService message.
- Configuration Option shall be set to the same value as in the OfferService message.

⌋

7.2.1.2.5 SubscribeEventgroup message

[SWS_CM_10377] Sending SOME/IP SubscribeEventgroup messages - initial

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00103, RS_SOMEIPSD_00006, RS_SOMEIPSD_00015

⌈The subscription to *at least one* `Event` (`ServiceInterface.event`) of an `Eventgroup` (`SomeipEventGroup`) by invoking the `Subscribe()` method (see [SWS_CM_00141]) of the specific `Event` class of the `ServiceProxy` class shall cause the sending of a SOME/IP SubscribeEventgroup messages in case there is no active subscription for the particular `Eventgroup` (either because there was no previous subscription to this particular `Eventgroup` or the TTL of every received SubscribeGroupAck message (see [SWS_CM_00206]) for the particular `Eventgroup` has already expired).

The subscription to *at least one* `Event` of an `Eventgroup` by invoking the `Subscribe()` method (see [SWS_CM_00141]) of the specific `Event` class of the `ServiceProxy` class shall *not* cause the sending of a SOME/IP SubscribeEventgroup messages in case there is an active subscription for the particular `Eventgroup` (because there was some previous subscription to this particular `Eventgroup` and the TTL of at least one received SubscribeGroupAck message (see [SWS_CM_00206]) for the particular `Eventgroup` has not yet expired).

The client shall explicitly request Initial Events for Field notifier according to [PRS_SOMEIPSD_00703] and [PRS_SOMEIPSD_00811].⌋

[SWS_CM_10381] Sending SOME/IP SubscribeEventgroup messages - renewal due to TTL expiry

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00103, RS_SOMEIPSD_00006, RS_SOMEIPSD_00015

⌈Upon reception of an OfferService message, a SubscribeEventgroup message shall be sent to refresh/renew the active subscription to the particular `Eventgroup` if the TTL of an active subscription for a particular `Eventgroup` has not yet expired and there is at least one active subscription for an Event of this Eventgroup.⌋

[SWS_CM_00051] Sending SOME/IP SubscribeEventgroup messages - renewal due to detected reboot

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00103, RS_SOMEIPSD_00015, RS_SOMEIPSD_00018

⌈If a reboot of a server for an active subscription for a particular Eventgroup has been detected (see [SWS_CM_00050]) and there is at least one active subscription for an Event of this Eventgroup, a SubscribeEventgroup message shall be sent to refresh the active subscription to the particular Eventgroup (see [PRS_SOMEIPSD_00449] and [PRS_SOMEIPSD_00704])⌋

[SWS_CM_00205] Content of SOME/IP SubscribeEventgroup message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00103, RS_SOMEIPSD_00006, RS_SOMEIPSD_00005, RS_SOMEIPSD_00015

⌈The fields in the SOME/IP SubscribeEventgroup message shall be as follows:

- The Type field and the TTL field shall be set to values suitable for a SubscribeEventgroup entry, which means that
  - The Type field shall be set to SubscribeEventgroup (see [PRS_SOMEIPSD_00386] for numerical value)
  - The TTL field shall be set to a value derived from Manifest, where the `RequiredSomeipServiceInstance` element aggregates the `SomeipRequiredEventGroup` in the role `requiredEventGroup`. The `SomeipRequiredEventGroup` aggregates the `sdClientEventGroupTimingConfig` where the `timeToLive` is defined.
- The Service ID shall be taken from the offer message.
- The Instance ID shall be taken from the offer message.
- Major Version shall be derived from the offer message.
- The Eventgroup ID field shall be derived from Manifest where the `RequiredSomeipServiceInstance` element aggregates the `SomeipRequiredEventGroup` in the role `requiredEventGroup`. The `SomeipRequiredEventGroup` contains the `eventGroup` reference to the `SomeipEventGroup` where the `eventGroupId` is defined.
- IPv4 Endpoint Option shall be sent if the offer message contains an IPv4 Endpoint Option. In this case the IPv4 Address sent in the IPv4 Endpoint Option of the SubscribeEventgroup message is configured in the Manifest where the `RequiredSomeipServiceInstance` element is mapped with the `ServiceInstanceToMachineMapping` to an `EthernetCommunicationConnector` of a `Machine`. The `EthernetCommunicationConnector` refers to a `NetworkEndpoint` in the role `unicastNetworkEndpoint` where an IPv4 Address is configured in the`Ipv4Configuration` element.
- IPv6 Endpoint Option shall be sent if the offer message contains an IPv6 Endpoint Option. In this case the IPv6 Address sent in the IPv6 Endpoint Option of

the SubscribeEventgroup message is configured in the Manifest where the RequiredSomeipServiceInstance element is mapped with the ServiceInstanceToMachineMapping to an EthernetCommunicationConnector of a Machine. The EthernetCommunicationConnector refers to a NetworkEndpoint in the role unicastNetworkEndpoint where an IPv6 Address is configured in the Ipv6Configuration element.

- The Transport Layer Protocol used in the IPv4 Endpoint option and/or IPv6 Endpoint option shall be derived from the Manifest where the SomeipEventGroup points either to SomeipEventDeployments where the transportProtocol is set to udp or to tcp. The SomeipServiceInstanceToMachineMapping element that maps the RequiredSomeipServiceInstance to an EthernetCommunicationConnector of a Machine the transport protocol and the port number.
  - The UDP port shall be derived from SomeipServiceInstanceToMachineMapping.udpPort. In case the port number (SomeipServiceInstanceToMachineMapping.udpPort) is configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.
  - The TCP port shall be derived from SomeipServiceInstanceToMachineMapping.tcpPort. In case the port number (SomeipServiceInstanceToMachineMapping.tcpPort) is configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that value shall be used.
- The InitialDataRequested flag shall be set to 1 for fields and to 0 for events.
- Reserved shall be set to 0.
- Counter shall be set to 0.

⌋

Note: In AUTOSAR Adaptive Platform (and ara::com) there are currently no use cases in having parallel subscribes by the same subscriber to the same eventgroup of the same service (, with the only difference being in the endpoint).

[SWS_CM_00206] SOME/IP SubscribeEventgroupAck message

Upstream requirements: RS_CM_00204, RS_SOMEIPSD_00015, RS_SOMEIPSD_00006, RS_SOMEIPSD_00002, RS_SOMEIPSD_00003, RS_SOMEIPSD_00005

⌈The fields in the SOME/IP SubscribeEventgroupAck message shall be as follows:

- The Type field and the TTL field shall be set to values suitable for a SubscribeEventgroupAck entry, which means that
  - The Type field shall be set to SubscribeEventgroupAck (see [PRS_SOMEIPSD_00391] for numerical value)

  - The TTL field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message (see [PRS_SOMEIPSD_00391])

- The Service ID field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- The Instance ID field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- The Major Version field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- The Eventgroup ID field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- The Multicast Option(s) shall be set in the following way
  - An IPv4 Multicast Option shall be derived from the Manifest if a `multicastThreshold` with a value greater 0 is defined for the `SomeipProvidedEventGroup` and a `ipv4MulticastIpAddress` is defined for the same `SomeipProvidedEventGroup`.
  - An IPv6 Multicast Option shall be derived from the Manifest if a `multicastThreshold` with a value greater 0 is defined for the `SomeipProvidedEventGroup` and a `ipv6MulticastIpAddress` is defined for the same `SomeipProvidedEventGroup`.
  - The Transport Layer Protocol shall be set to UDP. Only UDP is supported as transport layer protocol in the IPv4 Multicast Option and/or IPv6 Multicast Option.
  - The UDP Port shall be derived from the the Manifest where the `ProvidedSomeipServiceInstance` that aggregates the `SomeipProvidedEventGroup` has the `eventMulticastUdpPort` defined.
- The InitialDataRequested flag shall be set to 1 for fields and to 0 for events.
- Reserved shall be set to 0.
- Counter shall be set to 0.

⌋

[SWS_CM_00208] SOME/IP SubscribeEventgroupNack message

Upstream requirements: RS_CM_00204, RS_SOMEIPSD_00016, RS_SOMEIPSD_00006, RS_SOMEIPSD_00005

⌈The fields in the SOME/IP SubscribeEventgroupNack message shall be as follows:

- The Type field and the TTL field shall be set to values suitable for a SubscribeEventgroupNack entry, which means that

  - The type field shall be set to SubscribeEventgroupAck (see [PRS_SOMEIPSD_00394] for numerical value)
  - The TTL field shall be set to 0x000000 (see [PRS_SOMEIPSD_00394])
- The Service ID field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- The Instance ID field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- The Major Version field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- The Eventgroup ID field shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- The InitialDataRequested flag shall be set to 1 for fields and to 0 for events.
- Reserved shall be set to 0.
- Counter shall be set to 0.

⌋

7.2.1.2.6 StopSubscribe Eventgroup message

[SWS_CM_10378] Sending SOME/IP StopSubscribeEventgroup messages

Upstream requirements: RS_CM_00204, RS_CM_00104, RS_SOMEIPSD_00006, RS_SOMEIPSD_00005, RS_SOMEIPSD_00017

⌈Stopping the subscription of an `Event` (`ServiceInterface.event`) of an `Eventgroup` (`SomeipEventGroup`) by invoking the `Unsubscribe()` method (see [SWS_CM_00151]) of the specific `Event` class of the `ServiceProxy` class shall *not* cause the sending of a SOME/IP StopSubscribeEventgroup message if there are still active subscriptions for other `Event`s of the same `Eventgroup`.

Stopping the subscription of the *last* `Event` of an `Eventgroup` by invoking the `Unsubscribe()` method (see [SWS_CM_00151]) of the specific `Event` class of the `ServiceProxy` class shall cause the sending of a SOME/IP StopSubscribeEventgroup message.⌋

[SWS_CM_00207] Content of SOME/IP StopSubscribeEventgroup message

Upstream requirements: RS_CM_00204, RS_CM_00104, RS_SOMEIPSD_00006, RS_SOMEIPSD_00005, RS_SOMEIPSD_00017

⌈The fields in the SOME/IP StopSubscribeEventgroup message shall be as follows:

- The Type field and the TTL field shall be set to values suitable for a StopSubscribeEventgroup entry, which means that
  - The Type field shall be set to SubscribeEventgroup (see [PRS_SOMEIPSD_00386] for numerical value)
  - The TTL field shall be set to 0x000000 (see [PRS_SOMEIPSD_00389])
- The Service ID field shall be set to the same value as in the SubscribeEventgroup message.
- The Instance ID field shall be set to the same value as in the SubscribeEventgroup message.
- The Major Version field shall be set to the same value as in the SubscribeEventgroup message.
- The Eventgroup ID field shall be set to the same value as in the SubscribeEventgroup message.
- IPv4 Endpoint Option shall be set to the same value as in the SubscribeEventgroup message.
- IPv6 Endpoint Option shall be set to the same value as in the SubscribeEventgroup message.
- The InitialDataRequested flag shall be set to 1 for fields and to 0 for events.
- Reserved shall be set to 0.
- Counter shall be set to 0.

⌋

7.2.1.2.7 Link Loss

[SWS_CM_11375] Link loss on Client side ⌈In case the SOME/IP network binding detects a link loss on the client side, the SOME/IP service discovery shall react according to [PRS_SOMEIPSD_00752] (i.e., re-enter the initial wait phase once the link is up again and the service is still requested).⌋

[SWS_CM_11376] Link loss on Server side ⌈In case the SOME/IP network binding detects a link loss on the server side, the SOME/IP service discovery shall react according to [PRS_SOMEIPSD_00751] (i.e., re-enter the initial wait phase once the link is up again and the service is still requested).⌋

7.2.1.3 Accumulation of SOME/IP messages

[SWS_CM_10387] Data accumulation for UDP data transmission

Upstream requirements: RS_CM_00204

⌈To allow for the transmission of multiple SOME/IP event, method request and method response messages within a single UDP datagram, data accumulation for UDP data transmission shall be supported.⌋

[SWS_CM_10388] Enabling of data accumulation for UDP data transmission

Upstream requirements: RS_CM_00204

⌈Data accumulation for UDP data transmission over the `udpPort` and `unicastNetworkEndpoint` defined on the `EthernetCommunicationConnector` that is referenced by a `SomeipServiceInstanceToMachineMapping` shall be enabled if the attribute `SomeipServiceInstanceToMachineMapping.udpCollectionBufferSizeThreshold` is set to a value. In this case all event and method messages that are configured for data accumulation shall be aggregated in a buffer until a transmission trigger (see [SWS_CM_10389] and [SWS_CM_10390]) arrives and the data transmission starts.⌋

[SWS_CM_10389] Configuration of a data accumulation on a `ProvidedSomeipServiceInstance` for transmission over UDP

Upstream requirements: RS_CM_00204

⌈For a `ProvidedSomeipServiceInstance` all `method` responses and `event`s for which the `udpCollectionTrigger` is set to `never` shall be aggregated in a buffer until a trigger arrives that starts the data transmission.

The following trigger options shall be supported:

- a SOME/IP message needs to be transmitted for which the `udpCollectionTrigger` is set to `always`.
- the `udpCollectionBufferTimeout` is reached for one of the SOME/IP message already aggregated in the buffer.
- the buffer size defined by the attribute `udpCollectionBufferSizeThreshold` is reached.
- adding the `method` response or `event` to the buffer would lead to a message larger than the maximum possible size (e.g. MTU size). In this case the actual buffer shall be triggered before handling the new event or method response.

⌋

[SWS_CM_10390] Configuration of a data accumulation on a `RequiredSomeipServiceInstance` for transmission over UDP

Upstream requirements: RS_CM_00204

⌈For a `RequiredSomeipServiceInstance` all `method` requests for which the `udpCollectionTrigger` is set to `never` shall be aggregated in a buffer until a trigger arrives that starts the data transmission.

The following trigger options shall be supported:

- a SOME/IP message needs to be transmitted for which the `udpCollectionTrigger` is set to `always`.
- the `udpCollectionBufferTimeout` is reached for one of the SOME/IP message already aggregated in the buffer.
- the buffer size defined by the attribute `udpCollectionBufferSizeThreshold` is reached.
- adding the `method` request or `event` to the buffer would lead to a message larger than the maximum possible size (e.g. MTU size). In this case the actual buffer shall be triggered before handling the new event or method response.

⌋

In the following sections the term "sending of a SOME/IP message shall be requested" will be used to describe the fact that the sending of the message is requested but may be deferred due to data accumulation for UDP data transmission according to [SWS_CM_10388], [SWS_CM_10389], and [SWS_CM_10390].

7.2.1.4 Execution context of message reception actions

In the following sections the term "upon reception" will be used to describe the fact that certain actions (e.g, the deserialization of the payload according to [SWS_CM_10294]) will be performed at a point in time between the actual reception of a message and the call of the corresponding API (e.g., the `GetNewSamples()` method of the respective `Event` class). This specification deliberately does not explicitly state whether these actions will be performed in the context of message reception, in the context of the API call, or in a completely separate execution context to leave room for potential optimizations of a concrete ara::com implementation.

The only restriction imposed here refers to the execution context of the `EventReceiveHandler` (see [SWS_CM_00309]). – Executing the `EventReceiveHandler` in the context of the `GetNewSamples()` method is not allowed, since according to [SWS_CM_00181] the `EventReceiveHandler`shall use the `GetNewSamples()` method to access the retrieved event data.

[SWS_CM_11270] Selecting elements of the ServiceInterface for SecOC transmission

Upstream requirements: RS_CM_00801, RS_CM_00803

⌈It is possible to define which elements of the `ServiceInterface` of the particular `AdaptivePlatformServiceInstance` shall be securedby SecOC. The selection of `ServiceInterface` elements is done by the `ServiceInterfaceElementSecureComConfig`that is aggregated by `AdaptivePlatformServiceInstance`.

The following configuration in the `ServiceInterfaceElementSecureComConfig` is applicable:

- **Methods**
  The roles `methodCall` and `methodReturn` identify the `method(s)` that shall be sprotected by SecOC with the configuration settings that are available in the `ServiceInterfaceElementSecureComConfig` element.
- **Events**
  The role `event` identifies the `event(s)` that shall be protected by SecOC with the configuration settings that are availble in the `ServiceInterfaceElementSecureComConfig` element.
- **Fields**
  The roles `fieldNotifier`, `getterCall`, `getterReturn`, `setterCall` and `setterReturn` identify the `field` content that shall be protected by SecOC with the configuration settings that are available in the `ServiceInterfaceElementSecureComConfig` element.

⌋

7.2.1.5 Handling Events

[SWS_CM_10287] Conditions for sending of a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017

⌈The sending of a SOME/IP event message shall be requested by invoking the `Send()` ([SWS_CM_00162]) / `Send()` ([SWS_CM_90437]) method of the respective `Event` class

- If there is static service connection according to [SWS_CM_02201]
- If there is at least one active subscriber and the offer of the service containing the event has not been stopped (either because the TTL contained in the SOME/IP OfferService message (see [SWS_CM_00203]) has expired or because the `StopOfferService()` method (see [SWS_CM_00111]) of the `ServiceSkeleton` class has been called).

⌋

[SWS_CM_10288] Transport protocol for sending of a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00010

⌈The SOME/IP event message shall be transmitted using the transport protocol defined via the `SomeipServiceInterfaceDeployment.eventDeployment.transportProtocol` attribute (see [TPS_MANI_03050]).⌋

[SWS_CM_10289] Source of a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042

⌈The SOME/IP event message shall use the unicast IP address and port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00307] and [PRS_SOMEIPSD_00315]) of the SOME/IP OfferService message ([SWS_CM_00203]) or the server address which has been statically pre-configured by the static service connection according to [SWS_CM_02201] as source address and source port for the transmission.⌋

[SWS_CM_10290] Destination of a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042

⌈The SOME/IP event message shall use the multicast IP address and the port taken from the IPv4/v6 Multicast Option (see [PRS_SOMEIPSD_00326] and [PRS_SOMEIPSD_00333]) of the SOME/IP SubscribeEventgroupAck message (see [SWS_CM_00206]) or the client address which has been statically pre-configured by the static service connection according to [SWS_CM_02201] as destination address and destination port for the transmission if the threshold defined by the `multicastThreshold` attribute of the `SomeipProvidedEventGroup` that is aggregated by the `ProvidedSomeipServiceInstance` in the role `eventGroup` in the Manifest has been reached (see [PRS_SOMEIPSD_00134]). The SOME/IP event message shall use the unicast IP address and the port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00307] and [PRS_SOMEIPSD_00315]) of the SOME/IP SubscribeEventgroup message ([SWS_CM_00205]) as destination address and destination port for the transmission if this threshold has not been reached (see [PRS_SOMEIPSD_00134]). In case multiple Endpoint Options have been contained in the SOME/IP SubscribeEventgroup message, the one matching the selected transport protocol (see [SWS_CM_10289]) shall be used.⌋

[SWS_CM_10291] Content of the SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004

⌈The entries in the SOME/IP event message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `eventDeployment.eventId` by adding 0x8000 to the `eventDeployment.eventId`.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) is unused for event messages (according to [PRS_SOMEIP_00702]) and thus shall be set to 0x0000.
- In case of inactive Session Handling, see [SWS_CM_10240], the Session ID (see [PRS_SOMEIP_00703]) is unused for event messages and thus shall be set to 0x0000 (see [PRS_SOMEIP_00932]) and [PRS_SOMEIP_00925]).

  In case of active Session Handling, see [SWS_CM_10240], the Session ID is used for event messages and thus shall be incremented (with proper wrap around) upon every transmission of an event message (see [PRS_SOMEIP_00933], [PRS_SOMEIP_00934], [PRS_SOMEIP_00521], and [PRS_SOMEIP_00925]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `NOTIFICATION` (0x02).
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is unused for event messages and thus (according to [PRS_SOMEIP_00925]) shall be set to `E_OK` (0x00).
- The Payload shall contain the serialized payload (i.e., the serialized `VariableDataPrototype` composed by the `ServiceInterface` in role `event`) according to the SOME/IP serialization rules.

⌋

The serialization rules are explained in section 7.2.1.9.

[SWS_CM_10292] Checks for a received SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00008, RS_SOMEIP_00014

⌈Upon reception of a SOME/IP event message the following checks shall be conducted:

- Verify that the Protocol Version (see [PRS_SOMEIP_00052]) is set to 0x01.
- Use the Length (see [PRS_SOMEIP_00042]) being larger than 8 in combination with the Message type (see [PRS_SOMEIP_00055]) being set to `NOTIFICATION` to determine that the received SOME/IP message is actually an event.
- Use the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element in the Manifest to determine the right `ServiceInterface`.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches 0x8000 + the `eventId` attribute of one of the `SomeipEventDeployment`s of the `SomeipServiceInterfaceDeployment`.
- Verify that the Client ID (see [PRS_SOMEIP_00702]) is set to 0x0000.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches `SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion`.
- Verify that the Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is set to `E_OK` (0x00).

If any of the above checks fails the received SOME/IP event message shall be discarded and and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

[SWS_CM_10293] Identifying the right event

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00022

⌈Using the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element as well as the Method ID (see [PRS_SOMEIP_00245]) and 0x8000 + the `eventId` attribute of the `SomeipEventDeployment`s of the `SomeipServiceInterfaceDeployment`, the right event shall be identified.⌋

[SWS_CM_10379] Silently discarding SOME/IP event messages for unsubscribed events

Upstream requirements: RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004

⌈If the event identified according to [SWS_CM_10293] does not have an active subscription because the `Subscribe()` method (see [SWS_CM_00141]) of the specific

Event class of the ServiceProxy class has not been called, or the Unsubscribe() method (see [SWS_CM_00151]) of the specific Event class of the ServiceProxy class has been called, or the TTL of the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has expired, and if there is no static service connection according to [SWS_CM_02201], the received SOME/IP event message shall be silently discarded (i.e., [SWS_CM_10294], [SWS_CM_10295], and the receive handler shall not be invoked).⌋

[SWS_CM_10296] Invoke receive handler

Upstream requirements: RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004

⌈In case a receive handler was registered using the SetReceiveHandler() method (see [SWS_CM_00181]) of the respective Event class for the event determined according to [SWS_CM_10293] this registered receive handler shall be invoked when the corresponding Event is received.⌋

[SWS_CM_10294] Deserializing the payload

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00028

⌈Based on the event determined according to [SWS_CM_10293] the Payload of the SOME/IP event message (i.e., the serialized VariableDataPrototype composed by the ServiceInterface in role event) shall be de-serialized according to the SOME/IP serialization rules.⌋

The serialization rules are explained in section 7.2.1.9.

[SWS_CM_10295] Providing the received event data

Upstream requirements: RS_CM_00204, RS_CM_00202, RS_SOMEIP_00004

⌈The de-serialized payload containing the event data shall be provided via the GetNewSamples() method of the respective Event class for the event determined according to [SWS_CM_10293].⌋

7.2.1.6 Handling Triggers

[SWS_CM_10511] Conditions for sending of a SOME/IP trigger

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017

⌈The sending of a SOME/IP trigger shall be requested by invoking the Send() method of the respective Trigger class (see [SWS_CM_00721]).

The SOME/IP trigger shall be sent if at least one of the following conditions is fulfilled:

- If there is static service connection according to [SWS_CM_02201]
- If there is at least one active subscriber and the offer of the service containing the trigger has not been stopped (either because the TTL contained in the SOME/IP OfferService message (see [SWS_CM_00203]) has expired or because the `StopOfferService()` method (see [SWS_CM_00111]) of the `ServiceSkeleton` class has been called).

⌋

Please note that in the Manifest configuration the `SomeipServiceInterfaceDeployment.eventDeployment` is used to configure triggers in the same way as events. The only difference is that in case of a trigger the `SomeipEventDeployment` will reference the `Trigger` in the role `trigger`. Therefore the following specification items described in chapter 7.2.1.5 are also valid for `Trigger`s since a trigger defines a special kind of an event.

- [SWS_CM_10288]
- [SWS_CM_10289]
- [SWS_CM_10290]

[SWS_CM_10512] Content of the SOME/IP trigger

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004

⌈The entries in the SOME/IP trigger shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `eventDeployment.eventId` by adding 0x8000 to the `eventDeployment.eventId`.
- The Length (see [PRS_SOMEIP_00042]) shall be set to 8
- The Client ID (see [PRS_SOMEIP_00702]) is unused for triggers (according to [PRS_SOMEIP_00702]) and thus shall be set to 0x0000.
- In case of inactive Session Handling, see [SWS_CM_10240], the Session ID (see [PRS_SOMEIP_00703]) is unused for triggers and thus shall be set to 0x0000 (see [PRS_SOMEIP_00932]) and [PRS_SOMEIP_00925]).

  In case of active Session Handling, see [SWS_CM_10240], the Session ID is used for triggers and thus shall be incremented (with proper wrap around) upon every transmission of an trigger (see [PRS_SOMEIP_00933], [PRS_SOMEIP_00934], [PRS_SOMEIP_00521], and [PRS_SOMEIP_00925]).

- The Protocol Version (see [PRS_SOMEIP_00051]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceVersion.majorVersion.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to NOTIFICATION (0x02).
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is unused for triggers and thus (according to [PRS_SOMEIP_00925]) shall be set to E_OK (0x00).

⌋

[SWS_CM_10513] Checks for a received SOME/IP trigger

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00008, RS_SOMEIP_00014

⌈Upon reception of a SOME/IP trigger the following checks shall be conducted:

- Verify that the Protocol Version (see [PRS_SOMEIP_00052]) is set to 0x01.
- Use the Length (see [PRS_SOMEIP_00042]) being equal to 8 in combination with the Message type (see [PRS_SOMEIP_00055]) being set to NOTIFICATION to determine that the received SOME/IP message is actually a trigger.
- Use the Service ID (see [PRS_SOMEIP_00245]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches 0x8000 + the eventId attribute of one of the SomeipEventDeployments of the SomeipServiceInterfaceDeployment.
- Verify that the Client ID (see [PRS_SOMEIP_00702]) is set to 0x0000.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion.
- Verify that the Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is set to E_OK (0x00).

If any of the above checks fails the received SOME/IP trigger shall be discarded and and the incident shall be logged (if logging is enabled for the ara::com implementation).⌋

[SWS_CM_10514] Identifying the right trigger

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00022

⌈Using the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element as well as the Method ID (see [PRS_SOMEIP_00245]) and 0x8000 + the `eventId` attribute of the `SomeipEventDeployment`s of the `SomeipServiceInterfaceDeployment`, the right trigger shall be identified.⌋

[SWS_CM_10515] Silently discarding SOME/IP triggers for unsubscribed triggers

Upstream requirements: RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004

⌈If the trigger identified according to [SWS_CM_10514] does not have an active subscription, the received SOME/IP trigger shall be silently discarded (i.e., [SWS_CM_00226], and [SWS_CM_00250] shall *not* be performed).⌋

[SWS_CM_10516] Invoke receive handler

Upstream requirements: RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004

⌈In case a receive handler was registered using the `SetReceiveHandler()` method (see [SWS_CM_00250]) of the respective `Trigger` class for the trigger determined according to [SWS_CM_10514] this registered receive handler shall be invoked when the corresponding Trigger is received.⌋

[SWS_CM_10517] Failures in sending a SOME/IP trigger

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_CM_00004

⌈If the sending of the `SOME/IP` trigger fails locally (due to a network error which is notified to the `ara::com` implementation), the `ara::com` implementation shall return `kNetworkBindingFailure` in the Result of the `Send()` method of the respective Trigger class (see [SWS_CM_00721]).⌋

7.2.1.7 Handling Method Calls

[SWS_CM_10297] Conditions for sending of a SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007

⌈The sending of a SOME/IP request message shall be requested by invoking the `operator()` of the respective `Method` class (see [SWS_CM_00196]) if there is static service connection according to [SWS_CM_02201] or if the providing service instance has not stopped offering the service (either because the TTL contained in the SOME/IP OfferService message (see [SWS_CM_00203]) has expired or because the `StopOf-`

ferService() method (see [SWS_CM_00111]) of the ServiceSkeleton class has been called).⌋

[SWS_CM_10441] Failures in sending of a SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007

⌈If the sending of the SOME/IP request message fails locally (in a way which is notified to the ara::com implementation), the ara::com implementation shall make the ara::core::Future returned by operator() of the respective Method class (see [SWS_CM_00196]) ready according to [SWS_CM_10440].⌋

[SWS_CM_10298] Transport protocol for sending of a SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00010

⌈The SOME/IP request message shall be transmitted using the transport protocol defined by the attribute SomeipServiceInterfaceDeployment.methodDeployment.transportProtocol in the Manifest.⌋

[SWS_CM_10299] Source of a SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00010

⌈The SOME/IP request message shall use the unicast IP address defined in the Manifest by the Ipv4Configuration/Ipv6Configuration attribute of the NetworkEndpoint that is referenced (in role unicastNetworkEndpoint) by the EthernetCommunicationConnector of a Machine which in turn is mapped to the RequiredSomeipServiceInstance by means of a SomeipServiceInstanceToMachineMapping as source address for the transmission. The port number configured via udpPort shall be used to derive the source port for the transmission in case the selected transport protocol (see [SWS_CM_10298]) is UDP. If this port number is configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that port shall be used. The port number configured via tcpPort shall be used to derive the source port for the transmission in case the selected transport protocol (see [SWS_CM_10298]) is TCP. If this port number is configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that port shall be used.⌋

[SWS_CM_10300] Destination of a SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007

⌈The SOME/IP request message shall use the unicast IP address and port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00307] and [PRS_SOMEIPSD_00315]) of the SOME/IP OfferService message ([SWS_CM_00203]) or the server address which has been statically pre-configured by the static service connection according to [SWS_CM_02201] as destination address and destination port for

the transmission. In case multiple Endpoint Options have been contained in the SOME/IP OfferService message, the one matching the selected transport protocol (see [SWS_CM_10298]) shall be used.⌋

[SWS_CM_10301] Content of the SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00003, RS_SOMEIP_00012, RS_SOMEIP_00021, RS_SOMEIP_00025, RS_SOMEIP_00041

⌈The entries in the SOME/IP request message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `methodDeployment.methodId`.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) shall be set to a value that uniquely identifies the client within a `Machine`. - This may be achieved by dynamically generating unique client IDs upon construction of the `ServiceProxy`.
- The Session ID (see [PRS_SOMEIP_00703]) shall be set to 0x0001 for the first call of a particular method by a given client and shall be incremented by 1 after each call performed by this client for the respective method (see [PRS_SOMEIP_00533]). Once the Session ID reaches 0xFFFF, it shall wrap around and start with 0x0001 again (see [PRS_SOMEIP_00521]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `REQUEST_NO_RETURN` (0x01) in case the `ClientServerOperation` referenced by `methodDeployment.method` contains a `fireAndForget` attribute which is set to `true`. The Message Type shall be set to `REQUEST` (0x00) otherwise.
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is unused for request messages and thus (according to [PRS_SOMEIP_00920]) shall be set to `E_OK` (0x00).
- The Payload shall contain the serialized payload (i.e., the `ArgumentDataPrototype`s of the `ClientServerOperation` with `direction` set to `in` and

inout serialized according to their order) according to the SOME/IP serialization rules.

⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10302] Checks for a received SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00003, RS_SOMEIP_00019, RS_SOMEIP_00021, RS_SOMEIP_00008, RS_SOMEIP_00014

⌈Upon reception of a SOME/IP request message the following checks shall be conducted:

- Verify that the Protocol Version (see [PRS_SOMEIP_00052]) is set to 0x01.
- Verify that the Length (see [PRS_SOMEIP_00042]) is larger than 7.
- Use the Message Type (see [PRS_SOMEIP_00055]) which is set to either REQUEST_NO_RETURN (0x01) or REQUEST (0x00) to determine that the received SOME/IP message is actually a SOME/IP request message.
- Use the Service ID (see [PRS_SOMEIP_00245]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches the methodId attribute of one of the SomeipMethodDeployments of the SomeipServiceInterfaceDeployment.
- Verify that the Message Type (see [PRS_SOMEIP_00055]) is set to REQUEST_NO_RETURN (0x01) in case the the ClientServerOperation referenced by methodDeployment.method of the SomeipMethodDeployment with matching methodId attribute contains a fireAndForget attribute which is set to true. Verify that the Message Type is set to REQUEST (0x00) otherwise.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion.
- Verify that the Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is set to E_OK (0x00).

If any of the above checks fails the received SOME/IP request message shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation). In case of a received REQUEST message (see [PRS_SOMEIP_00055]), additionally, an ERROR message with return code set to either E_WRONG_PROTOCOL_VERSION,

E_UNKNOWN_SERVICE, E_WRONG_INTERFACE_VERSION, E_UNKNOWN_METHOD, or E_WRONG_MESSAGE_TYPE (see [PRS_SOMEIP_00191]) shall be sent to the requester, depending on the detected error.⌋

[SWS_CM_10303] Identifying the right method

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00021

⌈Using the Service ID (see [PRS_SOMEIP_00245]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element as well as the Method ID (see [PRS_SOMEIP_00245]) and the methodId attribute of the SomeipMethodDeployments of the SomeipServiceInterfaceDeployment, the right method shall be identified.⌋

[SWS_CM_10304] Deserializing the payload

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00028

⌈Based on the method determined according to [SWS_CM_10303] the Payload of the SOME/IP request message shall be de-serialized according to the SOME/IP serialization rules.⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10306] Invoke the method - event driven

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007

⌈In case a ara::com::MethodCallProcessingMode==kEvent or ara::com::MethodCallProcessingMode==kEventSingleThread has been passed to the constructor of the ServiceSkeleton (see [SWS_CM_00130]), the de-serialized payload containing the method data (i.e., method ID and input arguments) shall be used to invoke the service method (see [SWS_CM_00191]) identified according to [SWS_CM_10303] of the ServiceSkeleton class as a consequence to the reception of the SOME/IP request message.⌋

[SWS_CM_10307] Invoke the method - polling

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007

⌈In case a ara::com::MethodCallProcessingMode==kPoll has been passed to the constructor of the ServiceSkeleton (see [SWS_CM_00130]), the de-serialized payload containing the method data (i.e., method ID and input arguments) shall be used to invoke the service method (see [SWS_CM_00191]) identified according to [SWS_CM_10303] of the ServiceSkeleton class upon a call to the ProcessNextMethodCall() method (see [SWS_CM_00199]) of the ServiceSkeleton class.⌋

[SWS_CM_10447] Dealing with unmodelled ApApplicationErrors

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007

⌈If the service method (see [SWS_CM_00191]) returns an ApApplicationError different from the modeled ones (i.e., different from the ones referenced by the ClientServerOperation in role possibleApError or in role possibleApErrorSet.apApplicationError),treating this as a violation according to [SWS_CORE_00003]. No message shall be sent back to the client.⌋

[SWS_CM_10308] Conditions for sending of a SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007

⌈The sending of a SOME/IP response message shall be requested upon availability of a result of the ara::core::Future, which either contains a valid value or an ara::core::ErrorCode matching one of the possible ApApplicationErrors referenced by the ClientServerOperation in the role possibleApError or in role possibleApErrorSet.apApplicationError of the service method (see [SWS_CM_10306] and [SWS_CM_10307]) in case the Message Type of the corresponding SOME/IP request message was set to REQUEST (0x00).⌋

[SWS_CM_10309] Transport protocol for sending of a SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00010

⌈The SOME/IP response message shall be transmitted using the transport protocol defined by the attribute SomeipServiceInterfaceDeployment.methodDeployment.transportProtocol in the Manifest.⌋

[SWS_CM_10310] Source of a SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00010

⌈The SOME/IP response message shall use the unicast IP address defined in the Manifest by the Ipv4Configuration/Ipv6Configuration attribute of the NetworkEndpoint that is referenced (in role unicastNetworkEndpoint) by the EthernetCommunicationConnector of a Machine which in turn is mapped to the ProvidedSomeipServiceInstance by means of a SomeipServiceInstanceToMachineMapping as source address for the transmission. The port number configured via udpPort shall be used to derive the source port for the transmission in case the selected transport protocol (see [SWS_CM_10309]) is UDP. If this port number is configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that port shall be used. The port number configured via tcpPort shall be used to derive the source port for the transmission in case the selected transport protocol (see [SWS_CM_10309]) is TCP. If this port number is configured to 0, an *ephemeral* port shall be used. If the port number is configured to a value different from 0 exactly that port shall be used.⌋

[SWS_CM_10311] Destination of a SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007

⌈The SOME/IP response message shall use the unicast source IP address and the source port of the corresponding received SOME/IP request message (see [SWS_CM_10299]) as destination address and destination port for the transmission.⌋

[SWS_CM_10312] Content of the SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00003, RS_SOMEIP_00012, RS_SOMEIP_00021, RS_SOMEIP_00025, RS_SOMEIP_00041, RS_SOMEIP_00008

⌈The entries in the SOME/IP response message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `methodDeployment.methodId`.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) shall be copied from the corresponding SOME/IP request message (see [SWS_CM_10301]).
- The Session ID (see [PRS_SOMEIP_00703]) shall be copied from the corresponding SOME/IP request message (see [SWS_CM_10301]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `ERROR` (0x81) in case the `ClientServerOperation` returned one of the possible `ApApplicationError`s referenced by the `ClientServerOperation` in role `possibleApError` or in role `possibleApErrorSet.apApplicationError`[1]. The Message Type shall be set to `RESPONSE` (0x80) otherwise.
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) shall be set to (`ApApplicationError.errorCode` + 0x1F) & 0xFF in case the `ClientServerOperation` raised one of the possible `ApApplicationError`s

[1]Note that this is in fact an incompatibility with the AUTOSAR classic platform (i.e., in cases where an AUTOSAR adaptive platform server operates with an AUTOSAR classic platform client) which defines that a Message Type of `RESPONSE` (0x80) shall be used in case an `ApApplicationError` is raised.

referenced by the ClientServerOperation in role possibleApError or in role possibleApErrorSet.apApplicationError. The Return Code shall be set to E_OK (0x00) otherwise.

- The Payload shall contain the serialized payload according to the SOME/IP serialization rules. In case of NO raised ApApplicationError, the ArgumentDataPrototypes of the ClientServerOperation with direction set to inout and out shall be serialized according to their order. – otherwise in case of a raised ApApplicationError, which is represented as an ara::core::ErrorCode contained in the ara::core::Result, the payload shall contain the serialized application error according to [SWS_CM_10428].

⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10428] payload representing application error

Upstream requirements: RS_SOMEIP_00014

⌈A raised application error shall be represented by a SOME/IP union: The type field of the union shall be set to 0x01. The element of the union with type field set to 0x01 shall be a SOME/IP struct with the following elements in depicted order:

- an uint64 representing the ApApplicationErrorDomain.value, to which the raised ApApplicationError belongs (ApApplicationError.errorDomain).
- an int32 representing the ApApplicationError.errorCode, which is represented on binding level as Value().

Additionally, following SOME/IP Transformation property values for the ApApplicationError are hard coded:

- sizeOfUnionLengthField/=32bit
- sizeOfUnionTypeSelectorField/=8bit
- sizeOfStructLengthField/=16bit
- sizeOfStringLengthField/=16bit
- byte-Order=network-byte-order(big endian)
- TLV for struct=no
- alignment=no
- String encoding=UTF-8
- String BOM=true
- String null-termination=true

⌋

[SWS_CM_10313] Checks for a received SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00003, RS_SOMEIP_00012, RS_SOMEIP_00019, RS_SOMEIP_00021, RS_SOMEIP_00025, RS_SOMEIP_00041, RS_SOMEIP_00008, RS_SOMEIP_00014

⌈Upon reception of a SOME/IP response message the following checks shall be conducted:

- Verify that the Protocol Version (see [PRS_SOMEIP_00052]) is set to 0x01.
- Verify that the Length (see [PRS_SOMEIP_00042]) is larger than 7.
- Use the Message Type (see [PRS_SOMEIP_00055]) which is set to either `RESPONSE` (0x80) or `ERROR` (0x81) to determine that the received SOME/IP message is actually a SOME/IP response message or error response message.
- Use the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element in the Manifest to determine the right `ServiceInterface`.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches the `methodId` attribute of one of the `SomeipMethodDeployment`s of the `SomeipServiceInterfaceDeployment`.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches `SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion`.
- Verify that the Client ID (see [PRS_SOMEIP_00702]) matches the client from the corresponding SOME/IP request message (see [SWS_CM_10301]).
- The Session ID (see [PRS_SOMEIP_00703]) matches the client from the corresponding SOME/IP request message (see [SWS_CM_10301]).

If any of the above checks fails the received SOME/IP response message shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

[SWS_CM_10314] Identifying the right method

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00021

⌈Using the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element as well as the Method ID (see [PRS_SOMEIP_00245]) and the `methodId` attribute of the `SomeipMethodDeployment`s of the `SomeipServiceInterfaceDeployment`, the right method shall be identified.⌋

[SWS_CM_10315] Discarding orphaned responses

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213

⌈In case the method call has been canceled according to [SWS_CM_00194] in the mean time, the received response/error messages of the canceled methods shall be ignored.⌋

[SWS_CM_10357] Distinguishing errors from normal responses

Upstream requirements: RS_CM_00204, RS_SOMEIP_00008

⌈The Message Type (see [PRS_SOMEIP_00055]) and the Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) of the SOME/IP message shall be used to determine whether the received SOME/IP message is a

- normal response (Message Type set to `RESPONSE` (0x80) **and** Return Code set to 0x0)
- an error response (Message Type set to `ERROR` (0x81)
- an error response (Message Type set to `RESPONSE`(0x80) and Return Code set to a value different from 0x0)[2]

The further processing of a response shall be performed according to [SWS_CM_10316], [SWS_CM_10358], [SWS_CM_10429], [SWS_CM_10430] and [SWS_CM_10317].⌋

[SWS_CM_10316] Deserializing the payload - normal response messages

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00028

⌈Based on the method determined according to [SWS_CM_10314] the Payload of the response message shall be de-serialized according to the SOME/IP serialization rules. – Therefore the `ArgumentDataPrototype`s with `direction` set to `inout` and `out` shall be de-serialized according to their order.⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10442] Failures during deserialization of response messages

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00028

⌈In case of failures during deserialization of response messages, the `ara::com` implementation shall make the `ara::core::Future` returned by `operator`

[2]The additional case of SOME/IP response messages with a Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) set to a value different from 0x0 is in place for the sake of compatibility with the AUTOSAR classic platform (i.e., AUTOSAR adaptive platform client and AUTOSAR classic platform server) which defines that a Message Type of `RESPONSE` (0x80) shall be used even in case `ApApplicationError`s are raised.

() of the respective Method class (see [SWS_CM_00196]) ready according to [SWS_CM_10440].⌋

[SWS_CM_10358] Identifying the right application error in a message with Message Type set to RESPONSE (0x80)

Upstream requirements: RS_CM_00204, RS_SOMEIP_00008

⌈If the Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) contains a value larger than 0x1F the corresponding value of the ApApplicationError.errorCode attribute shall be determined by subtracting 0x1F from the Return Code value. Using this computed ApApplicationError.errorCode attribute value and the ApApplicationError.errorCode attribute of all ApApplicationErrors referenced in role possibleApError by the ClientServerOperation corresponding to the method determined according to [SWS_CM_10314], the right application error shall be identified.

If this computed ApApplicationError.errorCode attribute value does not match any of the ApApplicationError.errorCode attributes of all ApApplicationErrors referenced in role possibleApError or in role possibleApErrorSet.apApplicationError by the ClientServerOperation, the ara::core::Future returned by operator() of the respective Method class (see [SWS_CM_00196]) shall be made ready according to [SWS_CM_00048]. If the ReturnCode value is less than or equals to E_WRONG_MESSAGE_TYPE (see [PRS_SOMEIP_00191]) the ara::core::Future returned by operator() of the respective Method class (see [SWS_CM_00196]) shall be made ready according to [SWS_CM_10440]. If the ReturnCode value is equal to kUnspecifiedE2EError (see [SWS_CM_10474]), the ara::core::Future returned by operator() of the respective Method class (see [SWS_CM_00196]) shall be made ready according to [SWS_CM_00049].⌋

Note: This is for backward compatibility to old servers and Classic AUTOSAR systems using RESPONSE (0x80) even in case of application errors.

[SWS_CM_10429] Identifying the right application error in a message with Message Type set to ERROR (0x81)

Upstream requirements: RS_CM_00204, RS_SOMEIP_00008

⌈If the Message Type is set to ERROR (0x81) then the corresponding ApApplicationError shall be identified by de-serializing the Payload of the message according to the error payload format described in [SWS_CM_10428].⌋

[SWS_CM_10430] Handling invalid messages with Message Type set to ERROR (0x81)

Upstream requirements: RS_CM_00204, RS_SOMEIP_00008

⌈If the Message Type is set to ERROR (0x81), and either the contained payload does NOT comply with [SWS_CM_10428] or the application error identified by the de-serialized `ApApplicationErrorDomain.value` and `ApApplicationError.errorCode` is not referenced in role `possibleApError` or in role `possibleApErrorSet.apApplicationError` by the related `ClientServerOperation`, the `ara::core::Future` returned by `operator()` of the respective `Method` class (see [SWS_CM_00196]) shall be made ready according to [SWS_CM_00048].⌋

[SWS_CM_10317] Making the Future ready

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007, RS_SOMEIP_00008

⌈In order to make the `ara::core::Future` returned by `operator()` of the respective `Method` class (see [SWS_CM_00196]) ready, depending on the type or received message (see [SWS_CM_10357]) either the `set_value` operation (see [SWS_CORE_00345] and [SWS_CORE_00346]) or the `SetError` (see [SWS_CORE_00353]) operation of the `ara::core::Promise` corresponding to this `ara::core::Future` shall be invoked. This will unblock any blocking `get()`, `wait()`, `wait_for()`, and `wait_until()` calls that have been performed on this `ara::core::Future`. – The `set_value` operation shall be invoked in case of a received normal response message using the de-serialized payload according to [SWS_CM_10316] as an argument. The `SetError` operation shall be invoked in case of a received error response message using the determined application error according to [SWS_CM_10358] and [SWS_CM_10429] of type `ara::core::ErrorCode` as an argument.⌋

[SWS_CM_10318] Invoke the notification function

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007

⌈If a notification function has been registered with the `ara::core::Future`'s `then` method (see [SWS_CM_00197]), this notification function shall be invoked.⌋

7.2.1.8 Handling Fields

[SWS_CM_10319] Conditions for sending of a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_SOMEIP_00018

⌈The sending of a SOME/IP event message shall be requested in one of the following cases:

- By invoking the `Update()` method of the respective `Field` class (see [SWS_CM_00119])
- If the `ara::core::Future` returned by the `SetHandler` registered with `RegisterSetHandler()` (see [SWS_CM_00116]) becomes ready

The SOME/IP event message shall be sent if at least one of the following conditions is fulfilled:

- If there is static service connection according to [SWS_CM_02201] or
- If there is at least one active subscriber and the offer of the service containing the event has not been stopped (either because the TTL contained in the SOME/IP OfferService message (see [SWS_CM_00203]) has expired or because the `StopOfferService()` method (see [SWS_CM_00111]) of the `ServiceSkeleton` class has been called).

⌋

[SWS_CM_10320] Transport protocol for sending of a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00010

⌈The SOME/IP event message shall be transmitted using UDP if the threshold defined by the `multicastThreshold` attribute of the `SomeipProvidedEventGroup` that is aggregated by the `ProvidedSomeipServiceInstance` in the role `eventGroup` in the Manifest has been reached (see [PRS_SOMEIPSD_00134]). The SOME/IP event message shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.fieldDeployment.notifier.transportProtocol` in the Manifest if this threshold has not been reached (see [PRS_SOMEIPSD_00802]).⌋

[SWS_CM_10321] Source of a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042

⌈The source address and the source port of the SOME/IP event message shall set according to [SWS_CM_10289].⌋

[SWS_CM_10322] Destination of a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042

⌈The destination address and the destination port of the SOME/IP event message shall be set according to [SWS_CM_10290].⌋

[SWS_CM_10323] Content of the SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00009

⌈The entries in the SOME/IP event message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `eventDeployment.eventId` by adding 0x8000 to the `eventDeployment.eventId`.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) is unused for event messages (according to [PRS_SOMEIP_00702]) and thus shall be set to 0x0000.
- In case of inactive Session Handling, see [SWS_CM_10240], the Session ID (see [PRS_SOMEIP_00703]) is unused for event messages and thus shall be set to 0x0000 (see [PRS_SOMEIP_00932]) and [PRS_SOMEIP_00925]).

  In case of active Session Handling, see [SWS_CM_10240], the Session ID is used for event messages and thus shall be incremented (with proper wrap around) upon every transmission of an event message (see [PRS_SOMEIP_00933], [PRS_SOMEIP_00934], [PRS_SOMEIP_00521], and [PRS_SOMEIP_00925]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `NOTIFICATION` (0x02).
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is unused for event messages and thus (according to [PRS_SOMEIP_00925]) shall be set to `E_OK` (0x00).
- The Payload shall contain the serialized payload (i.e., the serialized `Field` composed by the `ServiceInterface` in role `field`) according to the SOME/IP serialization rules.

⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10324] Checks for a received SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00014

⌈Upon reception of a SOME/IP event message the checks defined in [SWS_CM_10292] shall be conducted. If any of the above checks fails the received SOME/IP event message shall be discarded and and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

[SWS_CM_10325] Identifying the right event

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00022

⌈Using the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element as well as the Method ID (see [PRS_SOMEIP_00245]) and 0x8000 + the `eventId` attribute of the `SomeipFieldDeployment.notifier`s of the `SomeipServiceInterfaceDeployment`, the right event shall be identified.⌋

[SWS_CM_10380] Silently discarding SOME/IP event messages for unsubscribed events

Upstream requirements: RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004, RS_SOMEIP_00009

⌈If the event identified according to [SWS_CM_10325] does not have an active subscription because the `Subscribe()` method (see [SWS_CM_00141]) of the specific `Field` class of the `ServiceProxy` class has not been called, or the `Unsubscribe()` method (see [SWS_CM_00151]) of the specific `Field` class of the `ServiceProxy` class has been called, or the TTL of the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has expired, the received SOME/IP event message shall be silently discarded (i.e., [SWS_CM_10326], [SWS_CM_10327], and [SWS_CM_10328] shall *not* be performed).⌋

[SWS_CM_10328] Invoke receive handler

Upstream requirements: RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004, RS_SOMEIP_00009

⌈In case a `ReceiveHandler` was registered using the `SetReceiveHandler()` method (see [SWS_CM_00181]) of the respective `Field` class for the event determined according to [SWS_CM_10325] this registered receive handler shall be invoked.⌋

[SWS_CM_10326] Deserializing the payload

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00028

⌈Based on the event determined according to [SWS_CM_10325] the Payload of the SOME/IP event message (i.e., the serialized `Field` composed by the `ServiceInterface` in role `field`) shall be de-serialized according to the SOME/IP serialization rules.⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10327] Providing the received event data

Upstream requirements: RS_CM_00204, RS_CM_00202, RS_SOMEIP_00004, RS_SOMEIP_00009

⌈The de-serialized payload containing the event data shall be provided via the `GetNewSamples()` method of the respective `Field` class for the event determined according to [SWS_CM_10325].⌋

[SWS_CM_10329] Conditions for sending of a SOME/IP request message

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈The sending of a SOME/IP request message shall be requested by invoking the `Set()` or `Get()` method of the respective `Field` class (see [SWS_CM_00112] and [SWS_CM_00113]) if the providing service instance has not stopped offering the service (either because the TTL contained in the SOME/IP OfferService message (see [SWS_CM_00203]) has expired or because the `StopOfferService()` method (see [SWS_CM_00111]) of the `ServiceSkeleton` class has been called).⌋

[SWS_CM_10443] Failures in sending of a SOME/IP request message

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈If the sending of the SOME/IP request message fails locally (in a way which is notified to the `ara::com` implementation), the `ara::com` implementation shall make the `ara::core::Future` returned by the `Set()` or `Get()` method of the respective `Field` class (see [SWS_CM_00112] and [SWS_CM_00113]) ready according to [SWS_CM_10440].⌋

[SWS_CM_10330] Transport protocol for sending of a SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010

⌈The SOME/IP request message for the `Set()` method shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.fieldDeployment.set.transportProtocol` in the Manifest. The SOME/IP request message for the `Get()` method shall be transmitted using the transport pro-

tocol defined by the attribute SomeipServiceInterfaceDeployment.fieldDeployment.get.transportProtocol respectively.⌋

[SWS_CM_10331] Source of a SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010

⌈The source address and the source port of the SOME/IP request message shall be set according to [SWS_CM_10299].⌋

[SWS_CM_10332] Destination of a SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈The destination address and the destination port of the SOME/IP request message shall be set according to [SWS_CM_10300].⌋

[SWS_CM_10333] Content of the SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_CM_00217, RS_CM_00218, RS_SOMEIP_00003, RS_SOMEIP_00012, RS_SOMEIP_00021, RS_SOMEIP_00025, RS_SOMEIP_00041

⌈The entries in the SOME/IP request message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Method ID (see [PRS_SOMEIP_00245]) for the Set() method shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the fieldDeployment.set.methodId. The Method ID for the Get() method shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the fieldDeployment.get.methodId.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) shall be set to a value that uniquely identifies the client within a Machine. – This may be achieved by dynamically generating unique client IDs upon construction of the ServiceProxy.
- The Session ID (see [PRS_SOMEIP_00703]) shall be set to 0x0001 for the first call of the particular method by a given client and shall be incremented by 1 after each call performed by this client for the respective method (see [PRS_SOMEIP_00533]). Once the Session ID reaches 0xFFFF, it shall wrap around and start with 0x0001 again (see [PRS_SOMEIP_00521]).

- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceVersion.majorVersion.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to REQUEST (0x00).
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is unused for request messages and thus (according to [PRS_SOMEIP_00920]) shall be set to E_OK (0x00).
- The Payload for the request message for the Set() method shall contain the serialized payload (i.e., the serialized Field composed by the ServiceInterface in role field) according to the SOME/IP serialization rules. The Payload for the request message for the Get() method will be empty.

⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10334] Checks for a received SOME/IP request message

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00003, RS_SOMEIP_00019, RS_SOMEIP_00021, RS_SOMEIP_00008, RS_SOMEIP_00014

⌈Upon reception of a SOME/IP request message the following checks shall be conducted:

- Verify that the Protocol Version (see [PRS_SOMEIP_00052]) is set to 0x01.
- Verify that the Length (see [PRS_SOMEIP_00042]) is larger than 7.
- Use the Message Type (see [PRS_SOMEIP_00055]) which is set to REQUEST (0x00) to determine that the received SOME/IP message is actually a SOME/IP request message.
- Use the Service ID (see [PRS_SOMEIP_00245]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches the methodId attribute of one of the SomeipMethodDeployments of the SomeipServiceInterfaceDeployment.
- Verify that the Message Type (see [PRS_SOMEIP_00055]) is set to REQUEST (0x00).
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion.

- Verify that the Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_-00191]) is set to `E_OK` (0x00).

If any of the above checks fails the received SOME/IP request message shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation). In case of a received REQUEST message (see [PRS_SOMEIP_00055]), additionally, an ERROR message with return code set to either `E_WRONG_PROTOCOL_VERSION`, `E_UNKNOWN_SERVICE`, `E_WRONG_INTERFACE_VERSION`, `E_UNKNOWN_METHOD`, or `E_WRONG_MESSAGE_TYPE` (see [PRS_SOMEIP_00191]) shall be sent to the requester, depending on the detected error.⌋

[SWS_CM_10335] Identifying the right method

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_-CM_00217, RS_CM_00218, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00021

⌈Using the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element as well as the Method ID (see [PRS_SOMEIP_00245]) and the `methodId` attribute of the `SomeipFieldDeployment.set`s and `SomeipFieldDeployment.get`s of the `SomeipServiceInterfaceDeployment`, the right method shall be identified.⌋

[SWS_CM_10336] Deserializing the payload

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00028

⌈Based on the method determined according to [SWS_CM_10335] the Payload of the SOME/IP request message shall be de-serialized according to the SOME/IP serialization rules.⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10338] Invoke the registered set/get handlers - event driven

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_-CM_00221, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈In case a `ara::com::MethodCallProcessingMode==kEvent` or `ara::com::MethodCallProcessingMode==kEventSingleThread` has been passed to the constructor of the `ServiceSkeleton` (see [SWS_CM_00130]), the de-serialized payload containing the method data (i.e., method ID and input arguments) shall be used to invoke a registered `SetHandler` resp. `GetHandler` (see [SWS_CM_00114] and [SWS_CM_00116]) of the `Field` class as a consequence to the reception of the SOME/IP request message.⌋

[SWS_CM_10339] Invoke the registered set/get handlers - polling

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈In case a `ara::com::MethodCallProcessingMode==kPoll` has been passed to the constructor of the `ServiceSkeleton` (see [SWS_CM_00130]), the deserialized payload containing the method data (i.e., method ID and input arguments) shall be used to invoke a registered `SetHandler` resp. `GetHandler` (see [SWS_CM_00114] and [SWS_CM_00116]) of the `Field` class upon a call to the `ProcessNextMethodCall()` method (see [SWS_CM_00199]) of the `ServiceSkeleton` class.⌋

[SWS_CM_10340] Conditions for sending of a SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈The sending of a SOME/IP response message shall be requested upon the return of a registered `SetHandler` resp. `GetHandler` (see [SWS_CM_00114] and [SWS_CM_00116]).⌋

[SWS_CM_10341] Transport protocol for sending of a SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010

⌈The SOME/IP response message for the `Set()` method shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.fieldDeployment.set.transportProtocol` in the Manifest. The SOME/IP response message for the `Get()` method shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.fieldDeployment.get.transportProtocol` respectively.⌋

[SWS_CM_10342] Source of a SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010

⌈The source address and the source port of the SOME/IP response message shall be set according to [SWS_CM_10310].⌋

[SWS_CM_10343] Destination of a SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈The destination address and the destination port of the SOME/IP response message shall be set according to [SWS_CM_10311].⌋

[SWS_CM_10344] Content of the SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00003, RS_SOMEIP_00012, RS_SOMEIP_00021, RS_SOMEIP_00025, RS_SOMEIP_00041, RS_SOMEIP_00008

⌈The entries in the SOME/IP response message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) for the `Set()` method shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `fieldDeployment.set.methodId`. The Method ID for the `Get()` method shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `fieldDeployment.get.methodId`.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) shall be copied from the corresponding SOME/IP request message (see [SWS_CM_10301]).
- The Session ID (see [PRS_SOMEIP_00703]) shall be copied from the corresponding SOME/IP request message (see [SWS_CM_10301]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `RESPONSE` (0x80).
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) shall be set to `E_OK` (0x00).
- The Payload shall contain the serialized payload (i.e., the serialized `Field` composed by the `ServiceInterface` in role `field`) which has either been provided by the value of the `ara::core::Future` returned by the registered `SetHandler` resp. `GetHandler` or obtained internally) according to the SOME/IP serialization rules.

⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10345] Checks for a received SOME/IP response message

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00003, RS_SOMEIP_00012, RS_SOMEIP_00019, RS_SOMEIP_00021, RS_SOMEIP_00025, RS_SOMEIP_00041, RS_SOMEIP_00008, RS_SOMEIP_00014

⌈Upon reception of a SOME/IP response message the checks defined in [SWS_CM_10313] shall be conducted. If any of the above checks fails the received SOME/IP event message shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

[SWS_CM_10346] Identifying the right method

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00021

⌈Using the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element as well as the Method ID (see [PRS_SOMEIP_00245]) and the `methodId` attribute of the `SomeipFieldDeployment.set`s and `SomeipFieldDeployment.get`s of the `SomeipServiceInterfaceDeployment`, the right method shall be identified.⌋

[SWS_CM_10347] Discarding orphaned responses

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213

⌈Orphaned responses shall be discarded according to [SWS_CM_10315].⌋

[SWS_CM_10348] Deserializing the payload

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00028

⌈Based on the method determined according to [SWS_CM_10346] the Payload of the SOME/IP response message shall be de-serialized according to the SOME/IP serialization rules.⌋

The SOME/IP serialization rules are explained in section 7.2.1.9.

[SWS_CM_10444] Failures during deserialization of response messages

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00028

⌈In case of failures during deserialization of response messages, the `ara::com` implementation shall make the `ara::core::Future` returned by the `Set()` or `Get()` method of the respective `Field` class (see [SWS_CM_00112] and [SWS_CM_00113]) ready according to [SWS_CM_10440].⌋

[SWS_CM_10349] Making the Future ready

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈In order to make the `ara::core::Future` returned by the `Set()` or `Get()` method of the respective `Field` class (see [SWS_CM_00113] and [SWS_CM_00112]) ready, the `set_value()` operation (see [SWS_CORE_00345] and [SWS_CORE_00346]) of the `ara::core::Promise` corresponding to this `ara::core::Future` shall be invoked using the de-serialized payload as an argument. This will unblock any blocking `Get()`, `wait()`, `wait_for()` and `wait_until()` calls that have been performed on this `ara::core::Future`.⌋

[SWS_CM_10350] Invoke the notification function

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007, RS_SOMEIP_00009

⌈Any registered notification function shall be invoked according to [SWS_CM_10318].⌋

[SWS_CM_10363] Failures in sending a SOME/IP event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_CM_00004

⌈If the sending of the `SOME/IP` event message generated by a field update fails locally (due to a network error which is notified to the ara::com implementation), the `ara::com` implementation shall return an error indicating "network binding failure" in the Result of the `Update()` method of the respective Field class (see [SWS_CM_00119]).⌋

7.2.1.9 Serialization of Payload

[SWS_CM_10034] Serialization of Payload

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00028

⌈The serialization of the payload shall be based on the definition of the `ServiceInterface` of the data.⌋

[SWS_CM_10259] Seralization Padding

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈After the serialized data of a variable data length `DataPrototype` a padding for alignment purposes shall be added for the configured alignment (see [SWS_CM_10260]) if the variable data length `DataPrototype` is not the last element in the serialized data stream.⌋

This requirement does not apply for the serialization of extensible structs and methods (see chapter 7.2.1.9.3).

[SWS_CM_10260] Setting the alignment for a variable data length data element

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If `SomeipDataPrototypeTransformationProps.someipTransformationProps.alignment` is set for a variable data length data element, the value of `SomeipDataPrototypeTransformationProps.someipTransformationProps.alignment` shall define the alignment. This requirement does not apply for the serialization of extensible structs and methods.⌋

(see chapter 7.2.1.9.3)

[SWS_CM_11262] Missing alignment for a variable data length data element

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If `SomeipDataPrototypeTransformationProps.someipTransformationProps.alignment` is not set for a variable data length data element, the value of `TransformationPropsToServiceInterfaceElementMapping.transformationProps.alignment` shall define the alignment. This requirement does not apply for the serialization of extensible structs and methods.⌋

(see chapter 7.2.1.9.3)

[SWS_CM_11263] Precedence of alignment settings for a variable data length data element

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If `SomeipDataPrototypeTransformationProps.someipTransformationProps.alignment` and `TransformationPropsToServiceInterfaceElementMapping.transformationProps.alignment` are both not set for a variable data length data element, no alignment shall be applied.⌋

[SWS_CM_10263] Padding for a fixed length data element

Upstream requirements: RS_CM_00201, RS_CM_00211

⌈After serialized fixed data length data elements, the SOME/IP network binding shall never add automatically a padding for alignment.⌋

Note:
If the following data element shall be aligned, a padding element of according size needs to be explicitly inserted into the `CppImplementationDataType`.

[SWS_CM_10037] Alignment calculation

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈Alignment shall always be calculated from start of SOME/IP message.⌋

This attribute defines the memory alignment. The SOME/IP network binding does not try to automatically align parameters but aligns as specified. The alignment is currently constraint to multiple of 1 Byte to simplify code generators.

SOME/IP payload should be placed in memory so that the SOME/IP payload is suitable aligned. For infotainment ECUs an alignment of 8 Bytes (i.e. 64 bits) should be achieved, for all ECU at least an alignment of 4 Bytes should be achieved. An efficient alignment is highly hardware dependent.

In the following the serialization of different parameters is specified.

7.2.1.9.1 Basic Data Types

[SWS_CM_10036] Serialization of supported primitive `StdCppImplementationDataType`s

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈

Type	Description	Size [bit]	Remark
boolean	TRUE/FALSE value	8	FALSE (0), TRUE (1)
std::uint8_t	unsigned Integer	8	
std::uint16_t	unsigned Integer	16	
std::uint32_t	unsigned Integer	32	
std::uint64_t	unsigned Integer	64	
std::int8_t	signed Integer	8	
std::int16_t	signed Integer	16	
std::int32_t	signed Integer	32	
std::int64_t	signed Integer	64	
float	floating point number	32	IEEE 754 binary32 (Single Precision)
double	floating point number	64	IEEE 754 binary64 (Double Precision)

⌋

Note: Primitive `StdCppImplementationDataType`s defined in [15].

The Byte Order is specified common for all parameters by `byteOrder` of `ApSomeipTransformationProps`.

7.2.1.9.2 Enumeration Data Types

[SWS_CM_10361] Serializing Enumeration Data Type

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈`Enumeration Data Type` shall be serialized according to [SWS_CM_10036] based on their underlying primitive `StdCppImplementationDataType` (i.e., the `Primitive Cpp Implementation Data Type` that is defined as the underlying type of the enumeration as defined in [SWS_LBAP_00027])⌋

7.2.1.9.3 Structured Data Types (structs)

[SWS_CM_10042] Serializing a struct Data Type

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈A `Structure Cpp Implementation Data Type` shall be serialized in order of depth-first traversal.⌋

The SOME/IP network binding doesn't automatically align parameters of a struct.

Insert reserved/padding elements into the AUTOSAR data type if needed for alignment, since the SOME/IP network binding shall not automatically add such padding.

So if for example a struct includes a std::uint8_t and a std::uint32_t, they are just written sequentially into the buffer. This means that there is no padding between the uint8 and the first byte of the std::uint32_t; therefore, the std::uint32_t might not be aligned. So the system designer has to consider to add padding elements to the data type to achieve the required alignment or set it globally.

Warning about unaligned structs or similar shall not be done in the SOME/IP network binding but only in the tool chain used to generate the SOME/IP network binding.

The SOME/IP network binding does not automatically insert dummy/padding elements.

SOME/IP allows to add a length field of 8, 16 or 32 bit in front of structs. The length field of a struct describes the number of bytes of the struct. This allows for extensible structs which allow better migration of interfaces.

[SWS_CM_00252] Missing size of length field for structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStructLengthField` is set to a value equal to 0, no length field shall be inserted in front of the serialized struct for which the `ApSomeipTrans-`

formationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10252]

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStructLengthField is set to a value greater 0, a length field shall be inserted in front of the serialized struct for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10268] Setting the size length field for structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is set this attribute shall define the byte order for the length field that shall be inserted in front of the serialized struct for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_00253] Default size of length field for structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfStructLengthField is set to a value equal to 0 and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStructLengthField is not set, no length field shall be inserted in front of the serialized struct for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_00254] Precedence when setting size of length field for structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfStructLengthField is set to a value greater 0 and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStructLengthField is not set, a length field shall be inserted in front of the serialized struct for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10269] Setting the byte order of the length field for structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder` is set and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder` is not set, the attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder` shall define the byte order for the length field that shall be inserted in front of the serialized struct for which the `ApSomeipTransformationProps` is defined via `SomeipDataPrototypeTransformationProps.someipTransformationProps`.⌋

[SWS_CM_00255] Default size of length field for structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfStructLengthField` is not set and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStructLengthField` is not set, no length field shall be inserted in front of the serialized struct.⌋

[SWS_CM_10270] Default byte order for the length field of structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder` is not set and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder` is not set, a byte order of `mostSignificantByteFirst` (i.e., big endian) shall be used for the length field that shall be inserted in front of the serialized associative struct.⌋

[SWS_CM_10253] Default data type for the length field of structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If `SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStructLengthField` defines the data type for the length field of a struct, the data shall be:

- *uint8* if `sizeOfStructLengthField` equals 1
- *uint16* if `sizeOfStructLengthField` equals 2
- *uint32* if `sizeOfStructLengthField` equals 4

⌋

[SWS_CM_00256] Default data type for the length field of structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If `TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfStructLengthField` defines the the data type for the length field of a struct, the data shall be:

- *uint8* if `sizeOfStructLengthField` equals 1
- *uint16* if `sizeOfStructLengthField` equals 2
- *uint32* if `sizeOfStructLengthField` equals 4

⌋

[SWS_CM_10218] Scope of length field value for structs

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The serializing SOME/IP network binding shall write the size (in bytes) of the serialized struct (without the size of the length field) into the length field of the struct.⌋

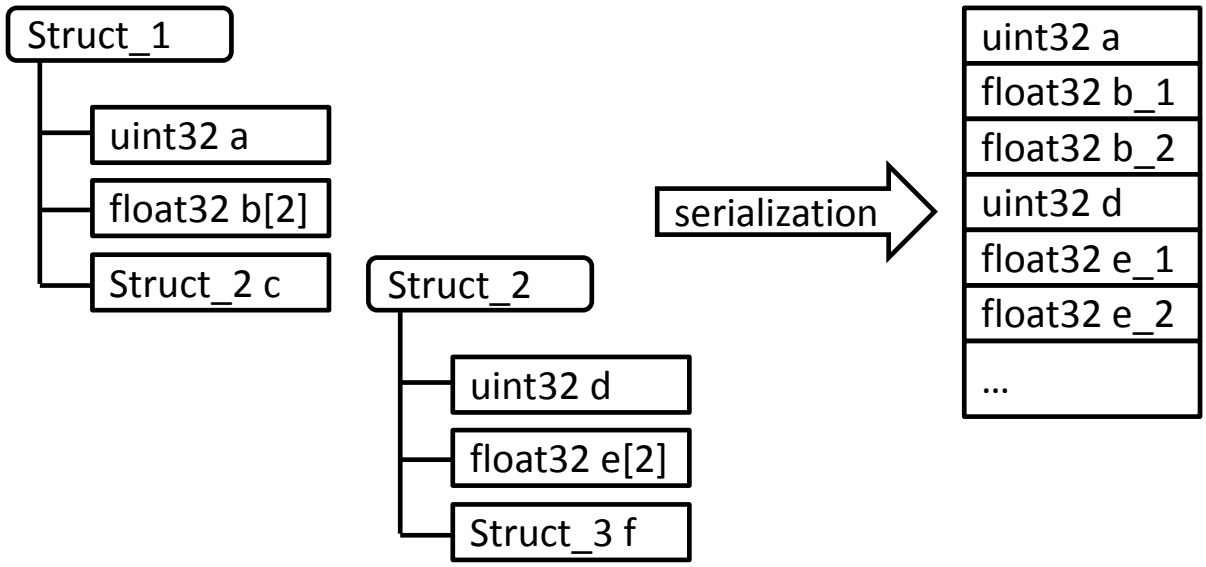

Figure 7.5: Serialization of Structs without Length Fields (Example)

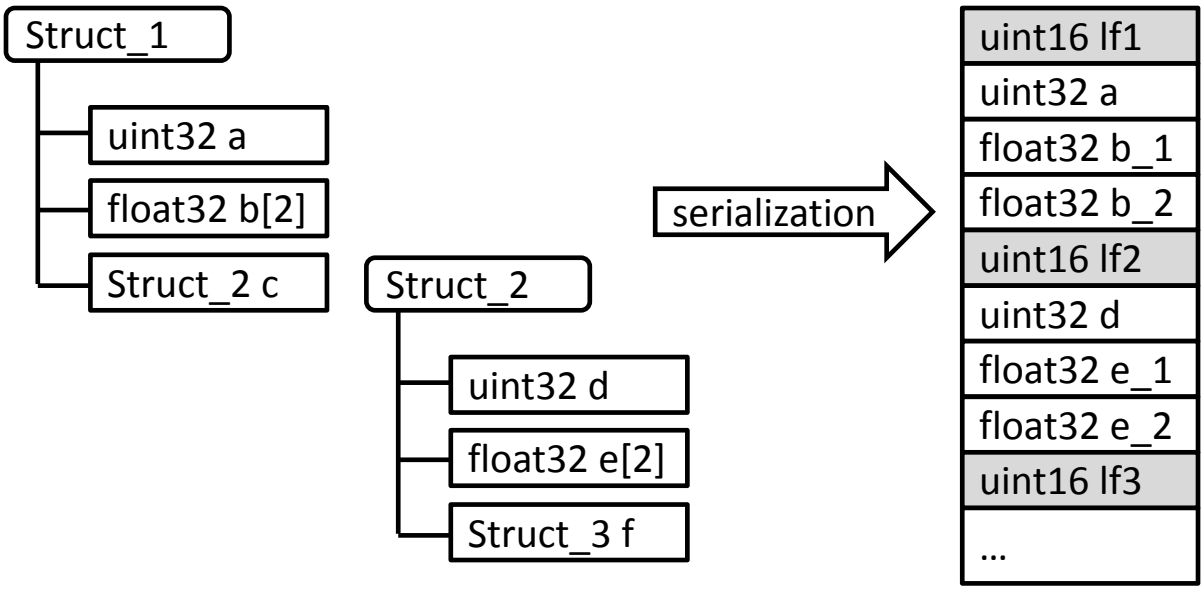

Figure 7.6: Serialization of Structs with Length Fields (Example)

[SWS_CM_01046] Definition of `tlvDataIdDefinition`

Upstream requirements: RS_CM_00205, RS_SOMEIP_00050

⌈Regarding the definition of `tlvDataIdDefinition` see [TPS_MANI_01097] and [constr_1594] for details.⌋

7.2.1.9.4 Structured Datatypes and Arguments with Identifier and optional Members

To achieve enhanced forward and backward compatibility, an additional Data ID can be added in front of struct members or method arguments. The receiver then can skip unknown members/arguments, i.e. where the Data ID is unknown. New members/arguments can be added at arbitrary positions when Data IDs are transferred in the serialized byte stream.

Structs are modeled in the Manifest using `CppImplementationDataType` of category `STRUCTURE` and members are represented by `CppImplementationDataTypeElement`s. Method arguments are represented by ArgumentDataPrototypes.

The assignment of Data IDs is modeled in the Manifest in the context of `TransformationPropsToServiceInterfaceElementMapping`. Refer to [5] for more details.

Moreover, the usage of Data IDs allows describing structs with optional members. Whether a member is optional or not, is defined in the Manifest using the attribute `CppImplementationDataTypeElement.isOptional`.

Whether an optional member is actually present in the struct or not, is to be determined during runtime. This is realized in the Adaptive Platform using the `ara::core::Optional` class template (see [3]).

In addition to the Data ID, a wire type encodes the datatype of the following member. Data ID and wire type are encoded in a so-called tag.

For more details, please refer to [4].

[SWS_CM_90443] Wire type for non-dynamic data types

Upstream requirements: RS_CM_00204

⌈If `TransformationPropsToServiceInterfaceElementMapping.transformationProps.isDynamicLengthFieldSize` is set to false or is not defined, the serializer shall use wire type 4 for serializing complex types and shall use the fixed size length fields. The size is defined in `TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfStructLengthField`, `sizeOfArrayLengthField` or `sizeOfStringLengthField`.⌋

[SWS_CM_90444] Wire type for dynamic data types

Upstream requirements: RS_CM_00204

⌈If `TransformationPropsToServiceInterfaceElementMapping.transformationProps.isDynamicLengthFieldSize` is set to true, the transformer shall use wire types 5,6,7 for serializing complex types and shall chose the size of the length field according to this wire type.⌋

[SWS_CM_90446] Data ID

Upstream requirements: RS_CM_00204

⌈If a Data ID is defined for an `ArgumentDataPrototype` or `CppImplementationDataType` by means of `TransformationPropsToServiceInterfaceElementMapping.TlvDataIdDefinition.id`, a tag shall be inserted in the serialized byte stream.⌋

Note: regarding existence of Data IDs, refer to [5].

Note: regarding existence of length field, refer to [4].

Rationale: The length field is required to skip unknown members/arguments during deserialization.

[SWS_CM_90451] Byte order for the length field of serialized structs

Upstream requirements: RS_CM_00204

⌈`TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder` shall define the byte order for the length field.⌋

[SWS_CM_90452] Default byte order for the length field of structs

Upstream requirements: RS_CM_00204

⌈If `TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder` is not defined, a byte order of mostSignificantByteFirst shall be used for the length field.⌋

For the length field of serialization of optional members see [PRS_SOMEIP_00229].

Regarding structure members and serialization examples, refer to [4].

7.2.1.9.5 Strings

[SWS_CM_10053] Strings encoding

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈Strings shall be encoded using Unicode and terminated with a "\0"-character.⌋

[SWS_CM_10054] Supported encoding

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136

⌈Different Unicode encoding shall be supported including UTF-8, UTF-16BE, and UTF-16LE. Since these encoding have a dynamic length of bytes per character, the maximum length in bytes is up to three times the length of characters in UTF-8 plus 1 Byte for the termination with a "\0" or two times the length of the characters in UTF-16 plus 2 Bytes for a "\0". UTF-8 character can be up to 6 bytes and an UTF-16 character can be up to 4 bytes.⌋

[SWS_CM_10285] Responsibility of proper string encoding

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136

⌈The application provides the string always in the UTF-8 encoding. The SOME/IP binding has to re-encode the data to the on-the-wire encoding that is configured by `ApSomeipTransformationProps.stringEncoding`.⌋

[SWS_CM_10055] UTF-16LE and UTF-16BE terminating bytes

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈UTF-16LE and UTF-16BE strings shall be zero terminated with a "\0" character. This means they shall end with (at least) two 0x00 Bytes.⌋

[SWS_CM_10056] UTF-16LE and UTF-16BE strings length

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈UTF-16LE and UTF-16BE strings shall have an even length.⌋

[SWS_CM_10057] Odd UTF-16LE and UTF-16BE string length

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈For UTF-16LE and UTF-16BE strings having an odd length the last byte shall be silently removed by the receiving SOME/IP network binding.⌋

[SWS_CM_10248] Odd UTF-16LE and UTF-16BE string length

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈In case of UTF-16LE and UTF-16BE strings having an odd length, after removal of the last byte, the two bytes before shall be 0x00 bytes (termination) for a string to be valid.⌋

[SWS_CM_10058] String start byte(BOM)

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈All strings shall always start with a Byte Order Mark (BOM).⌋

For the specification of BOM, see [16] and [17]. Please note that the BOM is used in the serialized strings to achieve compatibility with Unicode.

[SWS_CM_10459] Legacy string serialization

Status: OBSOLETE
Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The legacy string serialization shall be triggered if a Unicode is detected and attribute `ApSomeipTransformationProps.implementsLegacyStringSerialization` is true.⌋

[SWS_CM_10059] BOM checking by SOME/IP network binding implementation

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The receiving SOME/IP network binding implementation shall check the BOM and handle a missing BOM or a malformed BOM as an error by discarding the complete payload and logging the incident (if logging is enabled for the ara::com implementation).⌋

[SWS_CM_10060] BOM addition

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The BOM shall be added by the SOME/IP sending network binding implementation.⌋

[SWS_CM_10061] Supported encoding of CppImplementationDataType with category equal to STRING ⌈If a `CppImplementationDataType` with category equal to STRING is used in the context of a `ServiceInterface` then the encoding of this String DataType is UTF-8.⌋

This means that the CppImplementationDataType can only be mapped to an ApplicationDataType of category STRING where attribute swDataDefProps.swTextProps.baseType.baseTypeEncoding is set to the value UTF-8 as defined by [constr_5035]. If a CppImplementationDataType without an ApplicationDataType is used there is no formal description about the UTF-8 encoding in the ServiceInterface description.

According to SOME/IP serialized strings start with a length field of 8, 16 or 32 bit which precedes the actual string data. The value of this length field holds the length of the string including the BOM and any string termination in units of bytes.

[SWS_CM_10271] Default size of length field for strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStringLengthField is set to a value greater 0, a length field shall be inserted in front of the serialized string for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10272] Byte order of length field for strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is set this attribute shall define the byte order for the length field that shall be inserted in front of the serialized string for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10273] Size of length field for strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfStringLengthField is set to a value greater 0 and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStringLengthField is not set, a length field shall be inserted in front of the serialized struct for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10274] Setting byte order for the length field of strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder is set and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is not set, the attribute TransformationPropsToServiceInterfaceElementMapping.

transformationProps.byteOrder shall define the byte order for the length field that shall be inserted in front of the serialized string for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10275] Default size of length field for strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfStringLengthField is not set or set a value of 0 and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStringLengthField is not set or set to a value of 0, a length field of 4 bytes with the data type *uint32* shall be inserted in front of the serialized string.⌋

[SWS_CM_10276] Default byte order for the length field of strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder is not set and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is not set, a byte order of mostSignificantByteFirst (i.e., big endian) shall be used for the length field that shall be inserted in front of the serialized string.⌋

[SWS_CM_10277] Data type of the length field for strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfStringLengthField defines the the data type for the length field of a string, the data shall be:

- *uint8* if sizeOfStringLengthField equals 1
- *uint16* if sizeOfStringLengthField equals 2
- *uint32* if sizeOfStringLengthField equals 4

⌋

[SWS_CM_10278] Data type of the length field for strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfStringLengthField defines the the data type for the length field of a string, the data shall be:

- *uint8* if sizeOfStringLengthField equals 1

- *uint16* if `sizeOfStringLengthField` equals 2
- *uint32* if `sizeOfStringLengthField` equals 4

⌋

[SWS_CM_10245] Serialization of strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136

⌈Serialization of strings shall consist of the following steps:

1. Add the Length Field - The value of the length field shall be filled with the number of bytes needed for the string (i.e., the result of `ara::core::String::length()`), including the BOM and any string termination that needs to be added.
2. Appending BOM right after the length field according to the configured `ApSomeipTransformationProps.byteOrder`, if BOM is not already available in the first 3 (UTF-8) bytes of the to be serialized array containing the string. If the BOM is already present, simply copy the BOM into the output buffer.
3. Perform the re-encoding from UTF-8 to UTF-16 if the on-the-wire encoding is configured as UTF-16 by `ApSomeipTransformationProps.stringEncoding`. The re-encoding from UTF-8 to UTF-16BE shall be done if the configured `ApSomeipTransformationProps.byteOrder` is set to `mostSignificantByteFirst`. The re-encoding rom UTF-8 to to UTF-16LE shall be done if the configured `ApSomeipTransformationProps.byteOrder` is set to `mostSignificantByteLast`.
4. Copying the string data into the output buffer.
5. Termination of the string with `0x00`(UTF-8) or `0x0000` (UTF-16) if not terminated yet by appending `0x00`(UTF-8) or `0x0000` (UTF-16).

⌋

7.2.1.9.6 Vectors and arrays

SOME/IP supports arrays with static and dynamic length but there is no definition of vectors on this abstraction level. Therefore, vectors are mapped to arrays with dynamic length. The SOME/IP specification requires to add a length field of 8, 16 or 32 bit in front of data structures with dynamic length. The length field of arrays describes the total number of bytes. Note that this section uses only the term array which can also be used to realize vectors.

[SWS_CM_00270] Maximum number of vector elements

Upstream requirements: RS_SOMEIP_00037

⌈If a CppImplementationDataType of category VECTOR aggregates a templateArgument that defines the Allocator with the allocator reference (see [TPS_MANI_03186]), the maximum number of vector elements (according to [PRS_SOMEIP_00919]) shall be defined by the CppImplementationDataType.arraySize. If a CppImplementationDataType of category VECTOR does not aggregate a templateArgument that defines the Allocator with the allocator reference (see [TPS_MANI_03186]), the maximum number of vector elements is unbounded.⌋

[SWS_CM_00257] Missing size of array length field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField is set to a value equal to 0, no length field shall be inserted in front of the serialized array for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. – Note that omitting the length field by setting someipTransformationProps.sizeOfArrayLengthField to 0 is only allowed for arrays with static length (i.e., fixed length arrays) though (see also [constr_3447]).⌋

[SWS_CM_10256] Size of the length field for arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField is set to a value greater 0, a length field shall be inserted in front of the serialized array for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10279] Setting byte order for the length field of strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is set this attribute shall define the byte order for the length field that shall be inserted in front of the serialized array for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_00258] Default size of the length field for arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfArrayLengthField is set to a value equal to 0 and attribute SomeipDataPrototypeTransformationProps.someipTrans-

formationProps.sizeOfArrayLengthField is not set, no length field shall be inserted in front of the serialized array for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps. – Note that omitting the length field by setting someipTransformationProps.sizeOfArrayLengthField to 0 is only allowed for arrays with static length (i.e., fixed length arrays) though (see also [constr_3447]).⌋

[SWS_CM_00259] Setting size of the length field for arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfArrayLengthField is set to a value greater 0 and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField is not set, a length field shall be inserted in front of the serialized array for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10280] Setting the byte order for size of length field for arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder is set and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is not set, the attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder shall define the byte order for the length field that shall be inserted in front of the serialized array for which the ApSomeipTransformationProps is defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋

[SWS_CM_10258] Default size of the length field for arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfArrayLengthField is not set and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField is not set, a length field of 4 bytes with the data type *uint32* shall be inserted in front of the serialized array.⌋

[SWS_CM_10281] Byte order of length field for arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder is not set and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder is not

set, a byte order of mostSignificantByteFirst (i.e., big endian) shall be used for the length field that shall be inserted in front of the serialized array.⌋

[SWS_CM_10257] Datatype for the length field of arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField defines the the data type for the length field of a array, the data shall be:

- *uint8* if sizeOfArrayLengthField equals 1
- *uint16* if sizeOfArrayLengthField equals 2
- *uint32* if sizeOfArrayLengthField equals 4

⌋

[SWS_CM_00260] Datatype for the length field of arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfArrayLengthField defines the the data type for the length field of a array, the data shall be:

- *uint8* if sizeOfArrayLengthField equals 1
- *uint16* if sizeOfArrayLengthField equals 2
- *uint32* if sizeOfArrayLengthField equals 4

⌋

[SWS_CM_10076] Serializing arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈A array shall be serialized as the concatenation of the following elements:

- the length indicator which holds the length (in bytes) of the following array
- the array which contains the serialized elements of the array

where the size of the length field shall be determined as specified by ApSomeipTransformationProps.sizeOfArrayLengthField which applies to the array⌋

[SWS_CM_10234] Vector representation

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈A vector is represented in adaptive platform by a CppImplementationDataType with the category VECTOR. The payload is defined by a templateArgument that points with the templateType reference to the data type of elements that are con-

tained in the vector. Note that vectors are realized with dynamic sized arrays on SOME/IP level.⌋

[SWS_CM_10235] Array representation

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈An array is represented in adaptive platform by an `CppImplementationDataType` with the category `ARRAY`. The payload is defined by a `templateArgument` that points with the `templateType` reference to the data type of elements that are contained in the array. Note that `CppImplementationDataType` with the category `ARRAY` are realized with fixed length arrays on SOME/IP level.⌋

In case of nested arrays, the same scheme applies.

[SWS_CM_10222] Setting the size of the length field for arrays

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The serializing SOME/IP network binding shall write the size (in bytes) of the serialized array (without the size of the length field) into the length field.⌋

The layout of arrays with dynamic length is shown in 7.7 and Figure 7.8 where `L_1` and `L_2` denote the length in bytes. The serialization of one- and multi-dimensional dynamic length arrays is described in the next two subchapters.

One-dimensional

A one-dimensional array carries a number of elements of the same type.

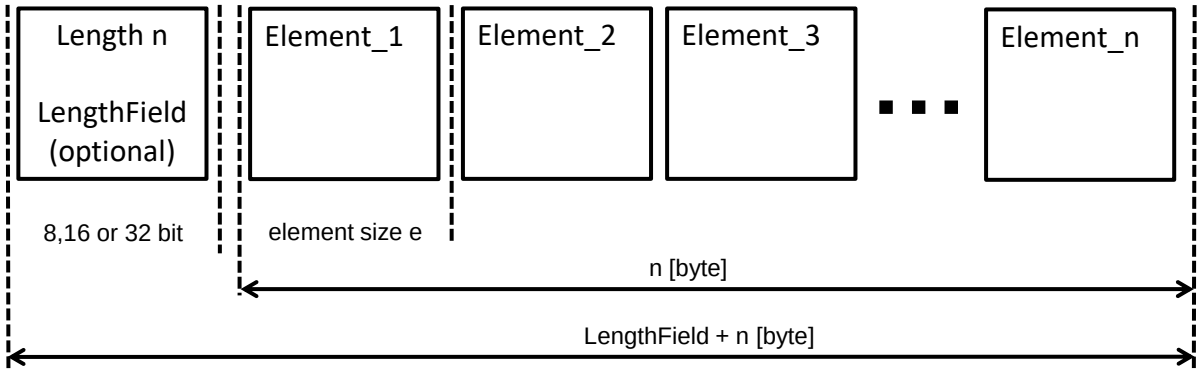

Figure 7.7: One-dimensional arrays (Example)

[SWS_CM_10070] Serializing one-dimentional array

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈A one-dimensional array shall be serialized by concatenating the arrays elements in order.⌋

Multi-dimensional

[SWS_CM_10072] Serializing multi-dimentional array

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The serialization of multi-dimensional arrays shall happen in depth-first order.⌋

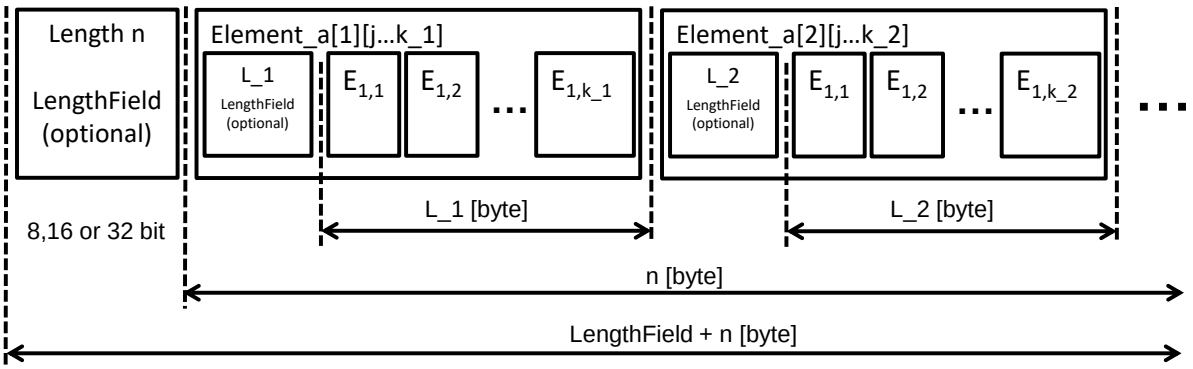

Figure 7.8: Multi-dimensional arrays (Example)

In case of multi-dimensional dynamic length arrays, each array (serialized as SOME/IP array) needs to have its own length field. See `L_1` and `L_2` in Figure 7.8.

7.2.1.9.7 Associative Maps

Associative map is modeled as `StdCppImplementationDataType` with `category` ASSOCIATIVE_MAP in the Manifest. As stated in the AUTOSAR Manifest Specification [5] the "natural" language binding in C++ for an associative map is `ara::core::Map<key_type,value_type>` where `key_type` is the data type used for the key of a map element and `value_type` is the data type for the value of a map element. Hereby `key_type` and `value_type` are derived from defined `CppTemplateArguments` aggregated by the `Associative Map Cpp Implementation Data Type`. Please see [SWS_LBAP_00023] for more details.

[SWS_CM_10261] Serialization of an associative map

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈As far as serialization is concerned the serialized representation of an associative map shall consist of the following parts without any intermediate padding:

- **Length field:** A length field describing the size of the associative map excluding the length field itself in units of bytes.
- **Elements:** The individual map elements themselves

⌋

[SWS_CM_10262] Insertion of an associative map length field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField` is set to a value greater 0, a length field shall be inserted in front of the serialized associative map for which the `ApSomeipTransformationProps` is defined via `SomeipDataPrototypeTransformationProps.someipTransformationProps`. – Note that omitting the length field by setting `someipTransformationProps.sizeOfArrayLengthField` to 0 is only allowed for arrays with static length (i.e., fixed length arrays) though (see also [constr_3447]).⌋

[SWS_CM_10282] Setting the byte order for size of the length field for associative maps

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder` is set this attribute shall define the byte order for the length field that shall be inserted in front of the serialized associative map for which the `ApSomeipTransformationProps` is defined via `SomeipDataPrototypeTransformationProps.someipTransformationProps`.⌋

[SWS_CM_00264] Setting the size of the length field for associative maps

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfArrayLengthField` is set to a value greater 0 and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField` is not set, a length field shall be inserted in front of the serialized associative map for which the `ApSomeipTransformationProps` is defined via `SomeipDataPrototypeTransformationProps.someipTransformationProps`. – Note that omitting the length field by setting `someipTransformationProps.sizeOfArrayLengthField` to 0 is only allowed for arrays with static length (i.e., fixed length arrays) though (see also [constr_3447]).⌋

[SWS_CM_10283] Setting the byte order for size of the length field for associative maps

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder` is set and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder` is not set, the attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder` shall define the byte order for the length field that shall be inserted in front of the serialized associative map for which the `ApSomeipTransformationProps` is defined via `SomeipDataPrototypeTransformationProps.someipTransformationProps`.⌋

[SWS_CM_10267] Insertion of an associative map length field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfArrayLengthField` is not set and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField` is not set, a length field of 4 bytes with the data type *uint32* shall be inserted in front of the serialized associative map.⌋

[SWS_CM_10284] Default byte order for size of the length field for associative maps

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute `TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder` is not set and attribute `SomeipDataPrototypeTransformationProps.someipTransformationProps.byteOrder` is not set, a byte order of `mostSignificantByteFirst` (i.e., big endian) shall be used for the length field that shall be inserted in front of the serialized associative map.⌋

[SWS_CM_10264] Size of the associative map length field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If `SomeipDataPrototypeTransformationProps.someipTransformationProps.sizeOfArrayLengthField` defines the the data type for the length field of an associative map, the data shall be:

- *uint8* if `sizeOfArrayLengthField` equals 1
- *uint16* if `sizeOfArrayLengthField` equals 2
- *uint32* if `sizeOfArrayLengthField` equals 4

⌋

[SWS_CM_00265] Datatype for the length field of associative maps

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If `TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfArrayLengthField` defines the the data type for the length field of an associative map, the data shall be:

- *uint8* if `sizeOfArrayLengthField` equals 1
- *uint16* if `sizeOfArrayLengthField` equals 2
- *uint32* if `sizeOfArrayLengthField` equals 4

⌋

[SWS_CM_10265] Serialization of associative map elements

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The individual elements of the associative map shall be serialized as a sequence of key-value pairs without any *additional* intermediate padding. Hereby the `key` attribute of an element shall be serialized first followed by the `value` attribute of this element.⌋

Table 7.1 illustrates the serialized form of an example map consisting of 3 elements where each element consists of a key-value pair of type `uint16` each. The `sizeOfArrayLengthField` is set to 4 bytes.

length field = 4 Bytes	
key0	value0
key1	value1
key2	value2

Table 7.1: Example of a serialized associative map

[SWS_CM_10266] Applicability of mandatory padding after variable length data elements

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈Any mandatory padding (see [TPS_MANI_03107] and [TPS_MANI_03073]) after variable length data elements (see [[TPS_MANI_03103], [TPS_MANI_03104], [TPS_MANI_03117] and [TPS_MANI_03105]) shall be applied after the serialized `key` attribute as well as after the `value` attribute in case the respective attributes is typed by a variable length data type. This requirement does not apply for the serialization of extensible structs and methods.⌋

(see chapter 7.2.1.9.3)

Note: Adhering to [SWS_CM_10266] is essential to ensure interoperability with the AUTOSAR classic platform where maps may be modelled as `ApplicationArrayDataType` with a `dynamicArraySizeProfile` of `VSA_LINEAR` where each array

element is an ApplicationRecordDataType of variable length and thus [TPS_-SYST_02126] applies to the individual ApplicationRecordElements.

7.2.1.9.8 Variants

A Variant (type-safe union) can contain different types of elements. For example, if one defines a Variant of type uint8 and type uint16, the Variant shall carry an element of uint8 or uint16. When using different types of elements the alignment of subsequent parameters may be distorted. To resolve this, padding might be needed.

[SWS_CM_10088] Default Serialization layout of Variants specified by the union data type in SOME/IP

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈

Length field (optional)
Type field
Element including padding [sizeof(padding) = length - sizeof(element)]

⌋

SOME/IP allows to add a length field of 8, 16 or 32 bit in front of unions (Variants). The length field of a union (Variant) describes the number of bytes in the union (Variant).

This allows the deserializing network binding to quickly calculate the position where the data after the union (Variant) begin in the serialized data stream. This gets necessary if the union (Variant) contains data which are larger than expected, for example if a struct was extended with appended new members and only the first "old" members are de-serialized by the SOME/IP network binding.

[SWS_CM_10254] Variant length field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If attribute sizeOfUnionLengthField of ApSomeipTransformationProps is set to a value greater 0, a length field shall be inserted in front of the serialized Variant for which the ApSomeipTransformationProps is defined.⌋

[SWS_CM_10255] Variant length field data type

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If ApSomeipTransformationProps.sizeOfUnionLengthField is present for a Variant specified the data type of the length field for the Variant shall be determined by the value of ApSomeipTransformationProps.sizeOfUnionLengthField:

- *uint8* if sizeOfUnionLengthField equals 1

- *uint16* if `sizeOfUnionLengthField` equals 2
- *uint32* if `sizeOfUnionLengthField` equals 4

⌋

For the length field of serialization of non-optional members see [PRS_SOMEIP_00126].

The type field describes the type of the element. For length of the type field see [PRS_SOMEIP_00127].

[SWS_CM_10251] Value of the variant type field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The value of the type field shall be set to the value which is returned by the `ara::core::Variant::index()` member function and incremented by 1.
Note: The `ara::core::Variant::index()` member function returns a zero-based index of the element hold in the Variant. A negative index represents a valueless Variant.⌋

[SWS_CM_10098] Possible values of the variant type field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈Possible values of the type field are defined by the elements of the Variant. The types are encoded in ascending order starting with 1 reusing the index encoding format of the Variant incremented by 1. The encoded value 0 is reserved for the NULL type - i.e. a valueless (empty) Variant.⌋

[SWS_CM_00052] NULL value of the variant type field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If a NULL type is received in a variant this shall be treated as an error by discarding the complete payload and logging the incident (if logging is enabled for the ara::com implementation).⌋

[SWS_CM_10099] Serialization of variant types

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈The element is serialized depending on the type in the type field. This also defines the length of the data. All bytes behind the data that are covered by the length, are padding. The deserializer shall skip the padding bytes by calculating the required number according to the formula given in [SWS_CM_10088].⌋

[SWS_CM_10230] Data type for size of union field

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If `ApSomeipTransformationProps.sizeOfUnionTypeSelectorField` is present for a specified Variant, the data type of the type selector field for the Variant shall be determined by the value of `ApSomeipTransformationProps.sizeOfUnionTypeSelectorField`:

- uint8 if `sizeOfUnionTypeSelectorField` equals 1
- uint16 if `sizeOfUnionTypeSelectorField` equals 2
- uint32 if `sizeOfUnionTypeSelectorField` equals 4

⌋

7.2.1.9.8.1 Example: Variant of uint8/uint16 both padded to 32 bit

In this example a length of the length field is specified as 32 bits. The Variant shall support a uint8 and a uint16 as elements. Both are padded to the 32 bit boundary (length=4 Bytes).

A uint8 will be serialized like this:

Length = 4 Bytes			
Type = 1			
uint8	Padding 0x00	Padding 0x00	Padding 0x00

A uint16 will be serialized like this:

Length = 4 Bytes		
Type = 2		
uint16	Padding 0x00	Padding 0x00

7.2.1.9.9 Segmentation of SOME/IP messages

[SWS_CM_10454] Event message segmentation

Upstream requirements: RS_SOMEIP_00051

⌈If the attribute `SomeipEventDeployment.maximumSegmentLength` is set to a value, and the data length is larger than `maximumSegmentLength`, `the SOME/IP event message` shall be transmitted/received using segmentation as described in [PRS_SOMEIP_00720] and following.⌋

[SWS_CM_99036] Event message separation time

Upstream requirements: RS_SOMEIP_00051

⌈If attribute `SomeipEventDeployment. separationTime` is set, and segmentation is activated for the corresponding SOME/IP event message according to [SWS_CM_10454], the segments shall be separated in time by this value.⌋

[SWS_CM_10455] Method request message segmentation

Upstream requirements: RS_SOMEIP_00051

⌈If the attribute `SomeipMethodDeployment.maximumSegmentLengthRequest` is set to a value, and the data length is larger than `maximumSegmentLengthRequest`, `the SOME/IP request message` shall be transmitted/received using segmentation as described in [PRS_SOMEIP_00720] and following.⌋

[SWS_CM_99037] Method request message separation time

Upstream requirements: RS_SOMEIP_00051

⌈If attribute `SomeipMethodDeployment. separationTimeRequest` is set, and segmentation is activated for the corresponding SOME/IP method request message according to [SWS_CM_10455], the segments shall be separated in time by this value.⌋

[SWS_CM_99038] Method response message segmentation

Upstream requirements: RS_SOMEIP_00051

⌈If the attribute `SomeipMethodDeployment.maximumSegmentLengthResponse` is set to a value, and the data length is larger than `maximumSegmentLengthResponse`, `the SOME/IP response message` shall be transmitted/received using segmentation as described in [PRS_SOMEIP_00720] and following.⌋

[SWS_CM_99039] Method response message separation time

Upstream requirements: RS_SOMEIP_00051

⌈If attribute `SomeipMethodDeployment. separationTimeResponse` is set, and segmentation is activated for the corresponding SOME/IP method response message according to [SWS_CM_99038], the segments shall be separated in time by this value.⌋

[SWS_CM_10456] Message segmentation for the get and set methods of fields

Upstream requirements: RS_SOMEIP_00051

⌈For the `get and set methods` aggregated by a `SomeipFieldDeployment` [SWS_CM_10455] shall apply. For the `notifier` aggregated by a `SomeipFieldDeployment` [SWS_CM_10454] shall apply.⌋

[SWS_CM_12023] Timeout time in message segmentation for Events

Upstream requirements: RS_SOMEIP_00010

⌈If the timeout time defined by `SomeipEventDeployment.segmentReceptionTimeoutTime` expires, the current assembly process shall be interrupted and and, if logging is enabled, the incident shall be logged.⌋

[SWS_CM_12024] Timeout time in message segmentation for Method Call

Upstream requirements: RS_SOMEIP_00010

⌈If the timeout time defined by `SomeipMethodDeployment.segmentReceptionTimeoutTimeRequest` expires, the current assembly process shall be interrupted and, if logging is enabled, the incident shall be logged.⌋

[SWS_CM_12025] Timeout time in message segmentation for Method Response

Upstream requirements: RS_SOMEIP_00010

⌈If the timeout time defined by `SomeipMethodDeployment.segmentReceptionTimeoutTimeResponse` expires, the current assembly process shall be interrupted and, if logging is enabled, the incident shall be logged.⌋

[SWS_CM_10457] Small messages segmentation

Upstream requirements: RS_SOMEIP_00051

⌈For messages that would fit into one segment no segmentation (i.e. no `TP-Header`) shall be applied.⌋

[SWS_CM_10445] SomeIpBurstTransmission

Upstream requirements: RS_SOMEIP_00051

⌈If parameter `SomeipEventDeployment.burstSize`, `SomeipMethodDeployment.burstSizeRequest` or `SomeipMethodDeployment.burstSizeResponse` is set to a value > 1 and the corresponding message is segmented no separationTime shall be applied for this number of segments. If not configured, `SeparationTime` will be applied between all frames.⌋

Note: If `burstSize` is set on receiver side it can be used to optimize buffer handling for reception of bursts.

7.2.1.10 De-serialization of Payload

[SWS_CM_10169] Missing parameters

Upstream requirements: RS_CM_00202

⌈To allow migration the deserialization shall ignore parameters attached to the end of previously known parameter list.⌋

This means: Parameters that were not defined in the `ServiceInterface` used to generate or parametrize the deserialization code but exist at the end of the serialized data will be ignored by the deserialization.

[SWS_CM_10016] Deserializing of exceeded unexpected data

Upstream requirements: RS_CM_00202

⌈If more data than expected shall be de-serialized, the unexpected data shall be discarded. The known fraction shall be considered.⌋

[SWS_CM_11411] Deserializing incomplete data on the skeleton side

Upstream requirements: RS_CM_00202

⌈If less data than expected shall be de-serialized on the skeleton side the data shall be discarded and the incident shall be logged. In case of a received REQUEST message (see [PRS_SOMEIP_00055]), additionally, an ERROR message with return code set to `E_MALFORMED_MESSAGE` (see [PRS_SOMEIP_00191]) shall be sent to the requester.⌋

[SWS_CM_11412] Deserializing incomplete data on the proxy side

Upstream requirements: RS_CM_00202

⌈If less data than expected shall be de-serialized on the proxy side and the data to be de-serialized does not belong to a Field or Event, the data shall be discarded and the incident shall be logged. In case of a received REQUEST message (see [PRS_SOMEIP_00055]), additionally, an ERROR message with return code set to `E_MALFORMED_MESSAGE` (see [PRS_SOMEIP_00191]) shall be sent to the requester.⌋

[SWS_CM_10017] Deserializing incomplete data on the proxy side belonging to a field and initValue defined

Upstream requirements: RS_CM_00202

⌈If less data than expected shall be de-serialized on the proxy side and the data to be de-serialized belong to a `Field` and the `initValue` is defined, the `initValue` shall be used as a substitute for the missing data.⌋

[SWS_CM_11413] Deserializing incomplete data on the proxy side belonging to a field and initValue not defined

Upstream requirements: RS_CM_00202

⌈If less data than expected shall be de-serialized on the proxy side and the data to be de-serialized belong to a `Field` and the `initValue` is not defined the data shall be discarded and the incident shall be logged. In case of a received REQUEST message (see [PRS_SOMEIP_00055]), additionally, an ERROR message with return code set to `E_MALFORMED_MESSAGE` (see [PRS_SOMEIP_00191]) shall be sent to the requester.⌋

[SWS_CM_12004] Deserializing incomplete data on the proxy side belonging to an event and `eventReceptionDefaultValue` is defined

Upstream requirements: RS_CM_00202

⌈If less data than expected shall be de-serialized on the proxy side and the data to be de-serialized belongs to an Event and the `eventReceptionDefaultValue` is defined, then the `eventReceptionDefaultValue` shall be used as a substitute for the missing data.⌋

[SWS_CM_12005] Deserializing incomplete data on the proxy side belonging to an event and `eventReceptionDefaultValue` is not defined

Upstream requirements: RS_CM_00202

⌈If less data than expected shall be de-serialized on the proxy side and the data to be de-serialized belongs to an Event and the `eventReceptionDefaultValue` is not defined, then the data shall be discarded and the incident shall be logged.⌋

7.2.1.10.1 Structured Data Types (structs)

[SWS_CM_10219] Length greater than expected struct length

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If the length is greater than the expected length of a struct (as specified in the data type definition) a de-serializing SOME/IP network binding shall only interpret the expected data and skip the unexpected.⌋

To determine the start of the next expected data following the skipped unexpected part, the SOME/IP network binding can use the supplied length information.

7.2.1.10.2 Structured Datatypes and Arguments with Identifier and optional Members

[SWS_CM_90445] A deserializer shall always be able to handle the wire types 4, 5, 6 and 7

Upstream requirements: RS_CM_00204

⌈A de-serializer shall always be able to handle the wire types 4, 5, 6 and 7 independent of the setting of `TransformationPropsToServiceInterfaceElementMapping.transformationProps.isDynamicLengthFieldSize`.⌋

7.2.1.10.3 Strings

[SWS_CM_10247] Deserialization of strings

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136

⌈Deserialization of strings shall consist of the following steps:

1. Check whether the string starts with a BOM. If not, the complete payload shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation). In case of a received REQUEST message (see [PRS_SOMEIP_00055]), additionally, an ERROR message with return code set to `E_MALFORMED_MESSAGE` (see [PRS_SOMEIP_00191]) shall be sent to the requester.
2. Check whether BOM has the same value as `ApSomeipTransformationProps.byteOrder`. If not, the complete payload shall be discarded and the incident shall be logged. In case of a received REQUEST message (see [PRS_SOMEIP_00055]), additionally, an ERROR message with return code set to `E_MALFORMED_MESSAGE` (see [PRS_SOMEIP_00191]) shall be sent to the requester.
3. Remove the BOM
4. Silently discard the last byte of the string in case of an UTF-16 string with odd length (in bytes)
5. Check whether the string terminates with `0x00` (UTF-8) or `0x0000` (UTF-16). If not, the complete payload shall be discarded and the incident shall be logged. In case of a received REQUEST message (see [PRS_SOMEIP_00055]), additionally, an ERROR message with return code set to `E_MALFORMED_MESSAGE` (see [PRS_SOMEIP_00191]) shall be sent to the requester.
6. Perform the re-encoding from UTF-16 to UTF-8 if the on-the-wire encoding is configured as UTF-16 by `ApSomeipTransformationProps.stringEncoding`. The re-encoding from UTF-16BE to UTF-8 shall be done if the configured

ApSomeipTransformationProps.byteOrder is set to mostSignificantByteFirst. The re-encoding from UTF-16LE to UTF-8 shall be done if the configured ApSomeipTransformationProps.byteOrder is set to mostSignificantByteLast.

7. Copy the string data (i.e., everything but the BOM and any string termination added during serialization).

⌋

7.2.1.10.4 Vectors and arrays

No further requirements considered for the deserialization.

7.2.1.10.5 Associative Maps

No further requirements considered for the deserialization.

7.2.1.10.6 Variants

[SWS_CM_10227] Length greater than expected Variant length

Upstream requirements: RS_CM_00201, RS_CM_00202, RS_CM_00211

⌈If the length is greater than the expected length of a Variant a deserializing SOME/IP network binding shall only interpret the expected data and skip the unexpected.⌋

To determine the start of the next expected data following the skipped unexpected part, the SOME/IP network binding can use the supplied length information.

7.2.1.10.7 Segmentation of SOME/IP messages

No further requirements considered for the deserialization.

7.2.1.11 Marker Interface

On the AUTOSAR adaptive platform there are use-cases for the utilization of a ServiceInterface that does not have any method, event, or field defined. In other words, the existence of a ServiceInterface by itself represents a valid semantics that has a value on its own.

A service instance that corresponds to such a ServiceInterface may be offered with the mere intention to signal that the ECU that provides the service instance is becoming ready for something. So the SOME/IP Service Discovery mechanism is used to indicate the readiness. But for the communication not SOME/IP but a different protocol will be used.

For example an ECU may indicate with a service offer that it is ready to being diagnosed. A tester could then take the existence of the offer as an indication to initiate a connection to the respective ECU.

[SWS_CM_10458] Handling of an ServiceInterface that does not contain any events, methods, or fields

Upstream requirements: RS_CM_00101

⌈If a `SomeipServiceInterfaceDeployment` is defined for a ServiceInterface that does not contain any events, methods, or fields and a `ProvidedSomeipServiceInstance` is defined in the `ServiceInstanceManifest` that points to the `SomeipServiceInterfaceDeployment` in the role serviceInterface then:

- the ServiceInterface shall be offered over SOME/IP as defined by [SWS_CM_00203] which means that the Endpoint Option shall include the IP-Address, Port Number and Protocol as defined by the `ProvidedSomeipServiceInstance`
- the Server shall not create a UDP/TCP socket and shall not bind any socket to the configured server address

⌋

7.2.2 Signal-Based Network binding

The applications on the adaptive platform communicate with each other in a service-oriented manner. When exchanging information with software components executed on an AUTOSAR classic platform which make use of signal-based communication, a conversion between this signal-based communication and the service-oriented communication needs to take place. Hereby the signals of a received signal-based communication is being made available as elements of a provided `ServiceInterface`. The signals of a sent signal-based communication are being made available as elements of a required `ServiceInterface`. The conversion between signal-based communication and service-oriented communication may be performed by a software component on an AUTOSAR classic platform gateway ECU or by an adaptive application on an AUTOSAR adaptive platform `Machine`.

There are several approaches how the signal-based information is made available at the adaptive AUTOSAR `Machine`:

- *Signal-Based SOME/IP* Network binding (see section 7.2.2.1)

- *Signal-Based Static* Network binding (see section 7.2.2.2)
- *Signal-Based IEEE1722 ACF* Network binding (see section 7.2.2.3).

7.2.2.1 Signal-Based SOME/IP Network binding

The `Signal-Based SOME/IP` network binding is currently a specialization of the `SOME/IP` network binding and many aspects of the `SOME/IP` network binding are re-used. Instead of replicating many specification items from the `SOME/IP` network binding the approach of this `Signal-Based SOME/IP` network binding chapter is to replicate the chapter structure. Specification items which are applicable to the `Signal-Based SOME/IP` network binding are just referenced, specification items which are NOT applicable to the `Signal-Based SOME/IP` network binding are explicitly excluded (via reference), and changed specification items are marked and the origin is referenced.

One major difference between the SOME/IP network binding and the `Signal-Based SOME/IP` network binding is the serialization technology. While the SOME/IP network binding only supports SOME/IP serialized payload the `Signal-Based SOME/IP` network binding supports the signal-based serialization of Classic platform COM-Stack as well as the SOME/IP serialization of payload (in order to support mixed use-cases).

[SWS_CM_11269] Definition of serialization technology

Upstream requirements: RS_CM_00204

⌈The serialization technology is defined by the attribute `SomeipEventDeployment.serializer`. If the attribute is set to `signalBased` then the signal-service-translation is responsible for the handling of the serialization. If the attribute is set to `someip` then the SOME/IP serializer is responsible for the handling of the serialization.⌋

See also chapter 7.2.2.1.7 and chapter 7.2.1.9.

In figure 7.9 an example of a mixed serialized service is illustrated. The event *x* is defined to use `someip serializer` while event *y* is defined to use `signalBased serializer`. Both are part of one service and share the service discovery and general event handling.

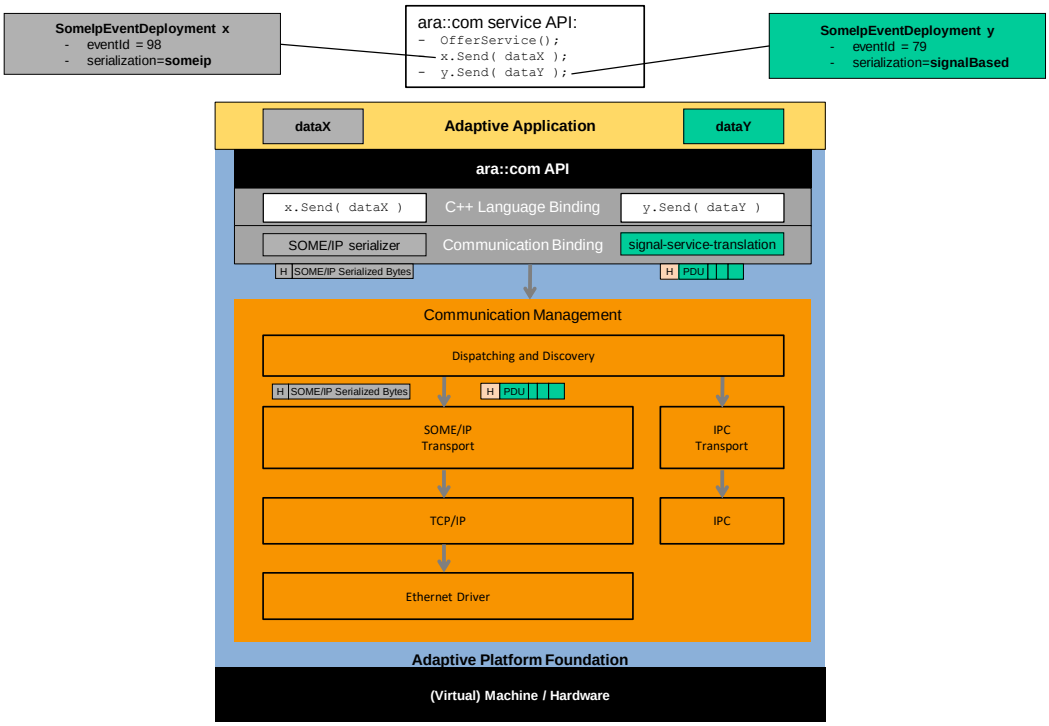

Figure 7.9: Example serialization settings

The modeling of the signal-based communication and the mapping between the individual elements of a `ServiceInterface` to the corresponding `ISignalTriggerings` is defined in the chapter “Signal-based communication” in [5].

[SWS_CM_10174] Mix of signal-based and SOME/IP communication

Upstream requirements: RS_CM_00204

⌈A combination of signal-based network binding and SOME/IP network binding shall be possible in a way to support the reception of a mix of signal-based communication and SOME/IP communication within a single UDP datagram or a single TCP stream on one UDP/TCP socket. Such a mix can occur when using [18] with enabled PDU-header option on the sender side.⌋

This allows to define the transport of messages from several services on the same socket, regardless of the serialization setting. Thus messages using the pure SOME/IP network binding can be transported together with messages using the signal-based network binding on the same socket.

Also one service - which consists of events with different serialization technologies (i.e. `someip` and `signalBased`) - shall be able to be transported on the same socket (this is covered by the signal-based network binding).

Based on [SWS_CM_10000]:

[SWS_CM_80001] Signal-based network binding shall implement SOME/IP and SOME/IP-SD

Upstream requirements: RS_CM_00204, RS_CM_00205, RS_CM_00004

⌈The signal-based network binding shall implement the SOME/IP Service Discovery Protocol defined in [6] and the SOME/IP Protocol defined in [4] (except for the serialization of signal-based payload).⌋

Length and Type fields shall always be in network byte order ([PRS_SOMEIP_00368] applies).

Based on [SWS_CM_10172]:

[SWS_CM_80003] Byte order for signal-based network binding with SOME/IP serialization

Upstream requirements: RS_SOMEIP_00026, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `someip` then
the byte order of the parameters inside the payload shall be defined by `byteOrder` of `ApSomeipTransformationProps`.⌋

[SWS_CM_80004] Byte order for signal-based network binding with signal-based serialization

Upstream requirements: RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `signalBased` then
the byte order of the parameters inside the payload shall be defined by the respective `packingByteOrder` of `ISignalToIPduMapping` and
by the `packingByteOrder` of `PduToFrameMapping`.⌋

[SWS_CM_10240] applies.

7.2.2.1.1 Service Discovery

The section 7.2.1.2 is fully applicable to the signal-based network binding.

7.2.2.1.2 Accumulation of messages

Based on [SWS_CM_10387]:

[SWS_CM_80017] Data accumulation for UDP data transmission

Upstream requirements: RS_CM_00004

⌈To allow for the transmission of multiple messages (SOME/IP event, SOME/IP method request, SOME/IP method response, signal-based event, and signal-based field notifier) within a single UDP datagram, data accumulation for UDP data transmission shall be supported.⌋

[SWS_CM_10388] applies.

Based on [SWS_CM_10389]:

[SWS_CM_80019] Configuration of a data accumulation on a `ProvidedSomeipServiceInstance` for transmission over UDP

Upstream requirements: RS_CM_00004

⌈For a `ProvidedSomeipServiceInstance` all `method` responses and `event`s for which the `udpCollectionTrigger` is set to `never` shall be aggregated in a buffer until a trigger arrives that starts the data transmission.

The following trigger options shall be supported:

- a message needs to be transmitted for which the `udpCollectionTrigger` is set to `always`.
- the `udpCollectionBufferTimeout` is reached for one of the message already aggregated in the buffer.
- the buffer size defined by the attribute `udpCollectionBufferSizeThreshold` is reached.
- adding the `method` response or `event` to the buffer would lead to a message larger than the maximum possible size (e.g. MTU size). In this case the actual buffer shall be triggered before handling the new event or method response.

⌋

Based on [SWS_CM_10390]:

[SWS_CM_80020] Configuration of a data accumulation on a `RequiredSomeipServiceInstance` for transmission over UDP

Upstream requirements: RS_CM_00004

⌈For a `RequiredSomeipServiceInstance` all `method` requests for which the `udpCollectionTrigger` is set to `never` shall be aggregated in a buffer until a trigger arrives that starts the data transmission.

The following trigger options shall be supported:

- a message needs to be transmitted for which the udpCollectionTrigger is set to always.
- the udpCollectionBufferTimeout is reached for one of the message already aggregated in the buffer.
- the buffer size defined by the attribute udpCollectionBufferSizeThreshold is reached.
- adding the method request or event to the buffer would lead to a message larger than the maximum possible size (e.g. MTU size). In this case the actual buffer shall be triggered before handling the new event or method response.

⌋

In the following sections the term "sending of a message shall be requested" will be used to describe the fact that the sending of the message is requested but may be deferred due to data accumulation for UDP data transmission according to [SWS_CM_10388], [SWS_CM_80019], and [SWS_CM_80020].

7.2.2.1.3 Handling Events

Based on [SWS_CM_10287]:

[SWS_CM_80021] Conditions for sending of an event message

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_CM_00004

⌈The sending of an event message shall be requested by invoking the

Send() ([SWS_CM_00162]) / Send() ([SWS_CM_90437]) method of the respective Event class if there is at least one active subscriber and the offer of the service containing the event has not been stopped (either because the TTL contained in the SOME/IP OfferService message (see [SWS_CM_00203]) has expired or because the StopOfferService() method (see [SWS_CM_00111]) of the ServiceSkeleton class has been called).⌋

Based on [SWS_CM_10288]:

[SWS_CM_80022] Transport protocol for sending of an event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00010, RS_CM_00004

⌈The event message shall be transmitted using UDP if the threshold defined by the multicastThreshold attribute of the SomeipProvidedEventGroup that is aggregated by the ProvidedSomeipServiceInstance in the role eventGroup in the Manifest has been reached (see [PRS_SOMEIPSD_00134]).

The event message shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.eventDeployment.transportProtocol` in the Manifest if this threshold has not been reached (see [PRS_SOMEIPSD_00802]).⌋

Based on [SWS_CM_10289]:

[SWS_CM_80023] Source of an event message

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042, RS_CM_00004

⌈The event message shall use the unicast IP address and port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00307] and [PRS_SOMEIPSD_00315]) of the SOME/IP OfferService message ([SWS_CM_00203]) as source address and source port for the transmission.⌋

Based on [SWS_CM_10290]:

[SWS_CM_80024] Destination of an event message

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042, RS_CM_00004

⌈The event message shall use the multicast IP address and the port taken from the IPv4/v6 Multicast Option (see [PRS_SOMEIPSD_00326] and [PRS_SOMEIPSD_00333]) of the SOME/IP SubscribeEventgroupAck message (see [SWS_CM_00206]) as destination address and destination port for the transmission if the threshold defined by the `multicastThreshold` attribute of the `SomeipProvidedEventGroup` that is aggregated by the `ProvidedSomeipServiceInstance` in the role `eventGroup` in the Manifest has been reached (see [PRS_SOMEIPSD_00134]). The event message shall use the unicast IP address and the port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00307] and [PRS_SOMEIPSD_00315]) of the SOME/IP SubscribeEventgroup message ([SWS_CM_00205]) as destination address and destination port for the transmission if this threshold has not been reached (see [PRS_SOMEIPSD_00134]). In case multiple Endpoint Options have been contained in the SOME/IP SubscribeEventgroup message, the one matching the selected transport protocol (see [SWS_CM_80023]) shall be used.⌋

Based on the `serviceInterfaceId` and `eventId` the respective event is determined. If the `serializer` is defined as `someip serializer` the SOME/IP event handling applies.

Based on [SWS_CM_10291]:

[SWS_CM_80025] Content of the SOME/IP serialized event message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `someip` then
the entries in the SOME/IP serialized event message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `eventDeployment.eventId` by adding 0x8000 to the `eventDeployment.eventId`.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) is unused for event messages (according to [PRS_SOMEIP_00702]) and thus shall be set to 0x0000.
- In case of inactive Session Handling, see [SWS_CM_10240], the Session ID (see [PRS_SOMEIP_00703]) is unused for event messages and thus shall be set to 0x0000 (see [PRS_SOMEIP_00932]) and [PRS_SOMEIP_00925]).

  In case of active Session Handling, see [SWS_CM_10240], the Session ID is used for event messages and thus shall be incremented (with proper wrap around) upon every transmission of an event message (see [PRS_SOMEIP_00933], [PRS_SOMEIP_00934], [PRS_SOMEIP_00521], and [PRS_SOMEIP_00925]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `NOTIFICATION` (0x02).
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is unused for event messages and thus (according to [PRS_SOMEIP_00925]) shall be set to `E_OK` (0x00).
- The Payload shall contain the serialized payload (i.e., the serialized `VariableDataPrototype` composed by the `ServiceInterface` in role `event`) according to the SOME/IP serialization rules.

⌋

If the serializer is defined as signalBased the signal-based event handling applies. As the message containing the signal-based payload is going to be routed to the Classic platform (without the SOME/IP Transformation) the header just contains the Message Id (i.e. ServiceID and Method ID) (see [SWS_CM_80026]).

[SWS_CM_80026] Content of the signal-based serialized event message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_CM_00004

⌈If SomeipEventDeployment.serializer is set to signalBased then
the entries in the signal-based event message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the eventDeployment.eventId by adding 0x8000 to the eventDeployment.eventId.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes
- The Payload shall contain the serialized payload (i.e., the serialized VariableDataPrototype composed by the ServiceInterface in role event) according to the signal-service-translation serialization rules defined in TPS-ManifestSpecification [5].

⌋

If the serializer is defined as someip serializer the SOME/IP event handling applies.

Based on [SWS_CM_10292]:

[SWS_CM_80027] Checks for a received SOME/IP serialized event message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00008, RS_SOMEIP_00014, RS_CM_00004

⌈If SomeipEventDeployment.serializer is set to someip then
upon reception of a SOME/IP serialized event message the following checks shall be conducted:

- Verify that the Protocol Version (see [PRS_SOMEIP_00052]) is set to 0x01.
- Use the Length being larger than 8 in combination with the Message type (see [PRS_SOMEIP_00055]) being set to NOTIFICATION to determine that the received SOME/IP message is actually an event.

- Use the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element in the Manifest to determine the right `ServiceInterface`.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches 0x8000 + `eventId` attribute of one of the `SomeipEventDeployment`s of the `SomeipServiceInterfaceDeployment` which have the attribute `SomeipEventDeployment.serializer` set to `someip`.
- Verify that the Client ID (see [PRS_SOMEIP_00702]) is set to 0x0000.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches `SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion`.
- Verify that the Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is set to `E_OK` (0x00).

If any of the above checks fails the received SOME/IP serialized event message shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

If the `serializer` is defined as `signalBased` the signal-based event handling applies. As the message containing the signal-based payload is coming from the Classic platform (without the `SOME/IP Transformation`) the header just contains the `Message Id` (i.e. `ServiceID` and `Method ID`) (see [SWS_CM_80028]).

[SWS_CM_80028] Checks for a received signal-based serialized event message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00008, RS_SOMEIP_00014, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `signalBased` then
upon reception of a signal-based serialized event message the following checks shall be conducted:

- Use the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element in the Manifest to determine the right `ServiceInterface`.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches 0x8000 + the `eventId` attribute of one of the `SomeipEventDeployment`s of the `SomeipServiceInterfaceDeployment` which have the attribute `SomeipEventDeployment.serializer` set to `signalBased`.
- Verify that the Length is larger than 0.

If any of the above checks fails the received signal-based event message shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

[SWS_CM_10293] applies.

Based on [SWS_CM_10379]:

[SWS_CM_80030] Silently discarding event messages for unsubscribed events

Upstream requirements: RS_CM_00203, RS_SOMEIP_00004, RS_CM_00004

⌈If the event identified according to [SWS_CM_10293] does not have an active subscription because the `Subscribe()` method (see [SWS_CM_00141]) of the specific `Event` class of the `ServiceProxy` class has not been called, or the `Unsubscribe()` method (see [SWS_CM_00151]) of the specific `Event` class of the `ServiceProxy` class has been called, or the TTL of the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has expired, then the received event message shall be silently discarded (i.e., [SWS_CM_80032], [SWS_CM_80033], [SWS_CM_10295], and [SWS_CM_10296] shall *not* be performed).⌋

[SWS_CM_10296] applies.

Based on [SWS_CM_10294]:

[SWS_CM_80032] Deserializing the SOME/IP serialized payload

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00028, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `someip` then based on the event determined according to [SWS_CM_10293] the Payload of the SOME/IP serialized event message (i.e., the serialized `VariableDataPrototype` composed by the `ServiceInterface` in role `event`) shall be de-serialized according to the SOME/IP serialization rules.⌋

Note: [SWS_CM_80032] supports the mix of `signal-based` and `SOME/IP` communication use case defined in [SWS_CM_10174].

[SWS_CM_80033] Deserializing the signal-based serialized payload

Upstream requirements: RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `signalBased` then based on the event determined according to [SWS_CM_10293] the Payload of the signal-based serialized event message (i.e., the serialized `VariableDataPrototype` composed by the `ServiceInterface` in role `event`) shall be de-serialized according to the signal-service-translation serialization rules defined in TPS-ManifestSpecification [5].⌋

[SWS_CM_10295] applies.

7.2.2.1.4 Handling Triggers

[SWS_CM_10518] Conditions for sending of a trigger

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_CM_00004

⌈The sending of an trigger shall be requested by invoking the `Send()` method of the respective `Trigger` class (see [SWS_CM_00721] if there is at least one active subscriber and the offer of the service containing the trigger has not been stopped (either because the TTL contained in the SOME/IP OfferService message (see [SWS_CM_00203]) has expired or because the `StopOfferService()` method (see [SWS_CM_00111]) of the `ServiceSkeleton` class has been called).⌋

Please note that in the Manifest configuration the `SomeipServiceInterfaceDeployment.eventDeployment` is used to configure triggers in the same way as events. The only difference is that in case of a trigger the `SomeipEventDeployment` will reference the `Trigger` in the role `trigger`. Therefore the following specification items described in chapter 7.2.2.1.3 are also valid for `Trigger`s since a trigger defines a special kind of an event.

- [SWS_CM_80022]
- [SWS_CM_80023]
- [SWS_CM_80024]

Based on the `serviceInterfaceId` and `eventId` the respective trigger is determined. If the `serializer` is defined as `someip serializer` the SOME/IP trigger handling applies.

[SWS_CM_10519] Content of the SOME/IP serialized trigger message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `someip` then
the entries in the SOME/IP serialized trigger message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `eventDeployment.eventId` by adding 0x8000 to the `eventDeployment.eventId`.
- The Length shall be set to 8
- The Client ID (see [PRS_SOMEIP_00702]) is unused for trigger (according to [PRS_SOMEIP_00702]) and thus shall be set to 0x0000.

- In case of inactive Session Handling, see [SWS_CM_10240], the Session ID (see [PRS_SOMEIP_00703]) is unused for trigger and thus shall be set to 0x0000 (see [PRS_SOMEIP_00932]) and [PRS_SOMEIP_00925]).

  In case of active Session Handling, see [SWS_CM_10240], the Session ID is used for trigger and thus shall be incremented (with proper wrap around) upon every transmission of an trigger (see [PRS_SOMEIP_00933], [PRS_SOMEIP_-00934], [PRS_SOMEIP_00521], and [PRS_SOMEIP_00925]).

- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceVersion.majorVersion`.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to `NOTIFICATION` (0x02).
- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is unused for trigger messages and thus (according to [PRS_SOMEIP_00925]) shall be set to `E_OK` (0x00).

⌋

If the `serializer` is defined as `signalBased` the signal-based trigger handling applies. As the message containing the signal-based payload is going to be routed to the Classic platform (without the `SOME/IP Transformation`) the header just contains the `Message Id` (i.e. `ServiceID` and `Method ID`) (see [SWS_CM_10520]).

[SWS_CM_10520] Content of the signal-based serialized trigger message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_-00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `signalBased` then the entries in the signal-based trigger shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `eventDeployment.eventId` by adding 0x8000 to the `eventDeployment.eventId`.
- The Length shall be set to 0.

⌋

If the serializer is defined as someip serializer the SOME/IP trigger handling applies.

[SWS_CM_10521] Checks for a received SOME/IP serialized trigger message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00008, RS_SOMEIP_00014, RS_CM_00004

⌈If SomeipEventDeployment.serializer is set to someip then
upon reception of a SOME/IP serialized trigger the following checks shall be conducted:

- Verify that the Protocol Version (see [PRS_SOMEIP_00052]) is set to 0x01.
- Use the Length being equal to 8 in combination with the Message type (see [PRS_SOMEIP_00055]) being set to NOTIFICATION to determine that the received SOME/IP message is actually a trigger.
- Use the Service ID (see [PRS_SOMEIP_00245]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches 0x8000 + the eventId attribute of one of the SomeipEventDeployments of the SomeipServiceInterfaceDeployment which have the attribute SomeipEventDeployment.serializer set to someip.
- Verify that the Client ID (see [PRS_SOMEIP_00702]) is set to 0x0000.
- Verify that the Interface Version (see [PRS_SOMEIP_00053]) matches SomeipServiceInterfaceDeployment.serviceInterfaceVersion.majorVersion.
- Verify that the Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is set to E_OK (0x00).

If any of the above checks fails the received SOME/IP serialized trigger shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation).⌋

If the serializer is defined as signalBased the signal-based trigger handling applies. As the message containing the signal-based payload is coming from the Classic platform (without the SOME/IP Transformation) the header just contains the Message Id (i.e. ServiceID and Method ID) (see [SWS_CM_10520]).

[SWS_CM_10522] Checks for a received signal-based serialized trigger

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00008, RS_SOMEIP_00014, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `signalBased` then upon reception of a signal-based serialized trigger the following checks shall be conducted:

- Use the Service ID (see [PRS_SOMEIP_00245]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element in the Manifest to determine the right `ServiceInterface`.
- Verify that the Method ID (see [PRS_SOMEIP_00245]) matches 0x8000 + the `eventId` attribute of one of the `SomeipEventDeployment`s of the `SomeipServiceInterfaceDeployment` which have the attribute `SomeipEventDeployment.serializer` set to `signalBased`.
- Verify that the Length is equal to 0.

If any of the above checks fails the received signal-based trigger shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

[SWS_CM_10514] applies.

[SWS_CM_10523] Silently discarding trigger for unsubscribed triggers

Upstream requirements: RS_CM_00203, RS_SOMEIP_00004, RS_CM_00004

⌈If the trigger identified according to [SWS_CM_10514] does not have an active subscription because the `Subscribe()` method (see [SWS_CM_00723]) of the specific `Trigger` class of the `ServiceProxy` class has not been called, or the `Unsubscribe()` method (see [SWS_CM_00810]) of the specific `Trigger` class of the `ServiceProxy` class has been called, or the TTL of the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has expired, then the received trigger shall be silently discarded (i.e., [SWS_CM_00226], and [SWS_CM_00250] shall *not* be performed).⌋

[SWS_CM_00250] applies.

7.2.2.1.5 Handling Method Calls

As the signal service translation does not apply to methods the handling is identical to the SOME/IP method serialization, see chapter 7.2.1.7.

7.2.2.1.6 Handling Fields

Based on [SWS_CM_10319]:

[SWS_CM_80063] Conditions for sending of an event message

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_SOMEIP_00018, RS_CM_00004

⌈The sending of an event message shall be requested by invoking the `Update()` method of the respective `Field` class (see [SWS_CM_00119]) or if the `ara::core::Future` returned by the `SetHandler` registered with `RegisterSetHandler()` (see [SWS_CM_00116]) becomes ready if there is at least one active subscriber and the offer of the service containing the event has not been stopped (either because the TTL contained in the SOME/IP OfferService message (see [SWS_CM_00203]) has expired or because the `StopOfferService()` method (see [SWS_CM_00111]) of the `ServiceSkeleton` class has been called).⌋

Based on [SWS_CM_10320]:

[SWS_CM_80064] Transport protocol for sending of an event message

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00010, RS_CM_00004

⌈The event message shall be transmitted using UDP if the threshold defined by the `multicastThreshold` attribute of the `SomeipProvidedEventGroup` that is aggregated by the `ProvidedSomeipServiceInstance` in the role `eventGroup` in the Manifest has been reached (see [PRS_SOMEIPSD_00134]).

The event message shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.fieldDeployment.notifier.transportProtocol` in the Manifest if this threshold has not been reached (see [PRS_SOMEIPSD_00802]).⌋

Based on [SWS_CM_10321]:

[SWS_CM_80065] Source of an event message

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042, RS_CM_00004

⌈The source address and the source port of the event message shall set according to [SWS_CM_80023].⌋

Based on [SWS_CM_10322]:

[SWS_CM_80066] Destination of an event message

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042, RS_CM_00004

⌈The destination address and the destination port of the event message shall be set according to [SWS_CM_80024].⌋

Based on the serviceInterfaceId and eventId the respective field notifier is determined. If the serializer is defined as someip serializer the SOME/IP serialized event handling applies.

Based on [SWS_CM_10323]:

[SWS_CM_80067] Content of the SOME/IP serialized event message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_CM_00004

⌈If SomeipEventDeployment.serializer is set to someip then
the entries in the SOME/IP serialized event message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the eventDeployment.eventId by adding 0x8000 to the eventDeployment.eventId.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes incremented by 8 (second part of the SOME/IP header that is covered by the Length)
- The Client ID (see [PRS_SOMEIP_00702]) is unused for event messages (according to [PRS_SOMEIP_00702]) and thus shall be set to 0x0000.
- In case of inactive Session Handling the Session ID (see [SWS_CM_10240]) the Session ID (see [PRS_SOMEIP_00703]) is unused for event messages and thus shall be set to 0x0000 (see [PRS_SOMEIP_00932]) and [PRS_SOMEIP_00925]).

  In case of active Session Handling, see [SWS_CM_10240], the Session ID is used for event messages and thus shall be incremented (with proper wrap around) upon every transmission of an event message (see [PRS_SOMEIP_00933], [PRS_SOMEIP_00934], [PRS_SOMEIP_00521], and [PRS_SOMEIP_00925]).
- The Protocol Version (see [PRS_SOMEIP_00052]) shall be set to 0x01.
- The Interface Version (see [PRS_SOMEIP_00053]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceVersion.majorVersion.
- The Message Type (see [PRS_SOMEIP_00055]) shall be set to NOTIFICATION (0x02).

- The Return Code (see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) is unused for event messages and thus (according to [PRS_SOMEIP_00925]) shall be set to `E_OK` (0x00).
- The Payload shall contain the serialized payload (i.e., the serialized `Field` composed by the `ServiceInterface` in role `field`) according to the SOME/IP serialization rules.

⌋

If the `serializer` is defined as `signalBased` the signal-based event handling applies. As the message containing the signal-based payload is going to be routed to the Classic platform (without the `SOME/IP Transformation`) the header just contains the `Message Id` (i.e. `ServiceID` and `Method ID`) (see [SWS_CM_80068]).

[SWS_CM_80068] Content of the signal-based serialized event message

Upstream requirements: RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `signalBased` then the entries in the signal-based serialized event message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00245]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `eventDeployment.eventId` by adding 0x8000 to the `eventDeployment.eventId`.
- The Length (see [PRS_SOMEIP_00042]) shall be set to the length of the serialized payload in units of bytes
- The Payload shall contain the serialized payload (i.e., the serialized `VariableDataPrototype` composed by the `ServiceInterface` in role `event`) according to the signal-service-translation serialization rules defined in TPS-ManifestSpecification [5].

⌋

If the `serializer` is defined as `someip serializer` the SOME/IP serialized event handling applies.

Based on [SWS_CM_10324]:

[SWS_CM_80069] Checks for a received SOME/IP serialized event message

Upstream requirements: RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00014, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `someip` then
upon reception of a SOME/IP serialized event message the checks defined in [SWS_CM_80027] shall be conducted.

If any of the above checks fails the received SOME/IP serialized event message shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

If the `serializer` is defined as `signalBased` the signal-based event handling applies. As the message containing the signal-based payload is coming from the Classic platform (without the `SOME/IP Transformation`) the header just contains the `Message Id` (i.e. `ServiceID` and `Method ID`) (see [SWS_CM_80070]).

[SWS_CM_80070] Checks for a received signal-based event message

Upstream requirements: RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `signalBased` then
upon reception of a signal-based event message the checks defined in [SWS_CM_80028] shall be conducted.

If any of the above checks fails the received signal-based event message shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

[SWS_CM_10325] applies.

Based on [SWS_CM_10380]:

[SWS_CM_80072] Silently discarding event messages for unsubscribed events

Upstream requirements: RS_CM_00203, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_CM_00004

⌈If the event identified according to [SWS_CM_10325] does not have an active subscription because the `Subscribe()` method (see [SWS_CM_00141]) of the specific `Field` class of the `ServiceProxy` class has not been called, or the `Unsubscribe()` method (see [SWS_CM_00151]) of the specific `Field` class of the `ServiceProxy` class has been called, or the TTL of the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has expired, then the received event message shall be silently discarded (i.e., [SWS_CM_80074], [SWS_CM_80101], [SWS_CM_10327], and [SWS_CM_10328] shall *not* be performed).⌋

[SWS_CM_10328] applies.

Based on [SWS_CM_10326]:

[SWS_CM_80074] Deserializing the SOME/IP serialized payload

Upstream requirements: RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00028, RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `someip` then
based on the event determined according to [SWS_CM_10325] the Payload of the SOME/IP serialized event message (i.e., the serialized `Field` composed by the `ServiceInterface` in role `field`) shall be de-serialized according to the SOME/IP serialization rules.⌋

Note: [SWS_CM_80074] supports the mix of `signal-based` and `SOME/IP` communication use case defined in [SWS_CM_10174].

[SWS_CM_80075] Deserializing the signal-based payload

Upstream requirements: RS_CM_00004

⌈If `SomeipEventDeployment.serializer` is set to `signalBased` then
based on the event determined according to [SWS_CM_10325] the Payload of the signal-based serialized event message (i.e., the serialized `Field` composed by the `ServiceInterface` in role `field`) shall be de-serialized according to the signal-service-translation serialization rules defined in TPS-ManifestSpecification [5].⌋

[SWS_CM_10327] applies.

[SWS_CM_10329] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10443] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10330] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10331] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10332] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10333] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10334] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10335] does not apply to Signal-Based SOME/IP network binding

[SWS_CM_10336] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10338] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10339] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10340] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10341] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10342] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10343] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10344] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10345] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10346] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10347] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10348] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10444] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10349] does not apply to Signal-Based SOME/IP network binding.

[SWS_CM_10350] applies.

[SWS_CM_10363] applies.

7.2.2.1.7 Serialization of Payload

The serialization technology is defined by the attribute `SomeipEventDeployment.serializer`. If the attribute is set to `signalBased` then the signal-service-translation is responsible for the handling of the serialization. If the attribute is set to `someip` then the SOME/IP serializer (see section 7.2.1.9) is responsible for the handling of the serialization.

[SWS_CM_80100] SOME/IP serialization of signal-based network binding

Upstream requirements: RS_CM_00004

⌈If the attribute `SomeipEventDeployment.serializer` is set to `someip` then the serialization of the payload shall be based on the SOME/IP serialization rules.⌋

Note: SOME/IP serialization rules are defined in section 7.2.1.9.

[SWS_CM_80101] Signal-based serialization

Upstream requirements: RS_CM_00004

⌈If the attribute `SomeipEventDeployment.serializer` is set to `signalBased` then
the serialization of the payload shall be based on the definition of the `ServiceInstanceToSignalMapping` defined for the signal-service-translation in TPS-ManifestSpecification [5].⌋

[SWS_CM_80102] Ignoring not mapped elements

Upstream requirements: RS_CM_00004, RS_CM_00202

⌈To allow migration the deserialization shall ignore signals which are not subject to ServiceInstanceToSignalMapping.⌋

7.2.2.1.8 De-Serialization of Payload

[SWS_CM_80103] Deserializing incomplete data belonging to a field

Upstream requirements: RS_CM_00004, RS_CM_00202

⌈If less data than expected shall be de-serialized and the data to be de-serialized belong to a Field, the initValue shall be used if it is defined. Otherwise the data shall be completely discarded and the incident shall be logged (if logging is enabled for the ara::com implementation).⌋

[SWS_CM_80104] Deserializing more data than expected

Upstream requirements: RS_CM_00004

⌈If more data than expected shall be de-serialized, the unexpected data shall be discarded. The known fraction shall be considered.⌋

[SWS_CM_12004] applies.

[SWS_CM_12005] applies.

7.2.2.2 Signal-Based Static Network binding

The Signal-Based Static network binding is enabled when a ServiceInstanceToSignalMapping refers to a ProvidedUserDefinedServiceInstance or RequiredUserDefinedServiceInstance of category SIGNAL-BASED_WITH_HEADER or SIGNALBASED_NO_HEADER.

Please note that there is currently no static *ara::com* API optimization defined, thus it is expected that the adaptive application, which interacts with a ServiceInterface, uses the same steps as in any other service oriented interaction (i.e. calling OfferService(), FindService(), Subscribe(), ...).

The general approach is:

For a ProvidedUserDefinedServiceInstance the connection is established in a UDP / TCP Server role.

For a RequiredUserDefinedServiceInstance the connection is established in a UDP / TCP Client role.

7.2.2.2.1 Service Discovery

[SWS_CM_80501] Mapping of Offer Service (Signal-Based Static network binding)

Upstream requirements: RS_CM_00004

⌈When instructed to *offer* a service instance which is mapped to a ProvidedUserDefinedServiceInstance of category SIGNALBASED_WITH_HEADER or SIGNALBASED_NO_HEADER, then the Signal-Based Static network binding shall create / use a socket for each entry in the remotePeers list. Each connection is defined by the localUdpPortNumber or localTcpPortNumber and one element out of the remotePeers list. If a connection with identical credentials already exists then this existing connection shall be used.

If a localUdpPortNumber is defined then each connection is created using the UDP protocol and bound to the listed remotePeers.

If a localTcpPortNumber is defined then each connection is created using the TCP protocol and is listening for client connections.⌋

[SWS_CM_80512] Mapping of Stop Offer Service (Signal-Based Static network binding)

Upstream requirements: RS_CM_00004

⌈When instructed to *stop offering* a service instance which is mapped to a ProvidedUserDefinedServiceInstance of category SIGNALBASED_WITH_HEADER or SIGNALBASED_NO_HEADER, then the Signal-Based Static network binding shall check:

- If this is the last service instance which uses the respective connection then this connection shall be closed.
- If there are still other service instance using this connection then the connection shall be kept open.

⌋

[SWS_CM_80502] Mapping of Find Service (Signal-Based Static network binding)

Upstream requirements: RS_CM_00004

⌈When instructed to *find* a service instance which is mapped to a RequiredUserDefinedServiceInstance of category SIGNALBASED_WITH_HEADER or SIGNALBASED_NO_HEADER, then the Signal-Based Static network binding shall immediately return a ara::com::ServiceHandleContainer with information about the static connection:

- localUdpPortNumber or localTcpPortNumber

- information about the EthernetCommunicationConnector (VLAN) where the connection shall be applied to
- a multicastIpAddress where the events will be consumed in case of multicast reception
- remotePeer information of the remote sender of the data (IP-Address and Port number)

⌋

[SWS_CM_80503] Mapping of Subscribe Service (Signal-Based Static network binding)

Upstream requirements: RS_CM_00004

⌈When instructed to *subscribe* to an event which is part of a RequiredUserDefinedServiceInstance of category SIGNALBASED_WITH_HEADER or SIGNALBASED_NO_HEADER, then the Signal-Based Static network binding shall:

If there is not already a socket connection established:

- TCP: use the information from the ara::com::ServiceHandleContainer create the socket and connect to the server.
- UDP: use the information from the ara::com::ServiceHandleContainer create the socket.

If there is already a socket connection established: use this socket connection.⌋

[SWS_CM_80513] Mapping of Unsubscribe Service (Signal-Based Static network binding)

Upstream requirements: RS_CM_00004

⌈When instructed to *un-subscribe* from an event which is part of a RequiredUserDefinedServiceInstance of category SIGNALBASED_WITH_HEADER or SIGNALBASED_NO_HEADER, then the Signal-Based Static network binding shall check:

- If this is the last service instance which uses the respective connection then this connection shall be closed.
- If there are still other service instance using this connection then the connection shall be kept open.

⌋

7.2.2.2.2 Accumulation of messages

[SWS_CM_80505] Data accumulation for UDP data transmission (`Signal-Based Static` network binding)

Upstream requirements: RS_CM_00004

⌈To allow for the transmission of multiple messages (signal-based events and signal-based field notifiers) within a single UDP datagram, data accumulation for UDP data transmission shall be supported.⌋

[SWS_CM_80504] Configuration of a data accumulation on a `RequiredUserDefinedServiceInstance` for transmission over UDP (`Signal-Based Static` network binding)

Upstream requirements: RS_CM_00004

⌈For a `ProvidedUserDefinedServiceInstance` of `category` `SIGNAL-BASED_WITH_HEADER` which has a `udpCollectionBufferSizeThreshold` > 0 defined, the events and field notifiers where `udpCollectionTrigger` is set to `never` shall be aggregated in a buffer until a trigger arrives that starts the data transmission.

The following trigger options shall be supported:

- a message needs to be transmitted for which the `udpCollectionTrigger` is set to `always`.
- the `udpCollectionBufferTimeout` is reached for one of the messages already aggregated in the buffer.
- the buffer size defined by the attribute `udpCollectionBufferSizeThreshold` is reached.
- adding the event of field notifier to the buffer would lead to a message larger than the maximum possible size (e.g. MTU size). In this case the actual buffer shall be triggered before handling the new event or field notifier.

⌋

7.2.2.2.3 Handling Events

[SWS_CM_80506] `Arbitrary Message Header` usage for `Signal-Based Static` network binding messages

Upstream requirements: RS_CM_00004

⌈If a `ProvidedUserDefinedServiceInstance` or `RequiredUserDefinedServiceInstance` of `category` `SIGNALBASED_WITH_HEADER` is defined then each

message shall have an `Arbitrary Message Header` (see [TPS_Manifest]) defined. This message header is composed of a 32 bit wide Message ID field and 32 bit wide Message Length field. Both encoded in big endian.

The the signal based payload is appended (the Message Length field is used to determine how long the payload is in bytes).⌋

[SWS_CM_80507] No header option for `Signal-Based Static` network binding messages

Upstream requirements: RS_CM_00004

⌈If a `ProvidedUserDefinedServiceInstance` or `RequiredUserDefinedServiceInstance` of `category` `SIGNALBASED_NO_HEADER` is defined then there is no header information standardized and thus the signal based payload is the only content of the message.⌋

7.2.2.2.4 Handling Method Calls

[SWS_CM_80508] No method support for `Signal-Based Static` network binding

Upstream requirements: RS_CM_00004

⌈The `Signal-Based Static` network binding does not support methods.⌋

7.2.2.2.5 Handling Fields

[SWS_CM_80509] Only field notifier support for `Signal-Based Static` network binding

Upstream requirements: RS_CM_00004

⌈The `Signal-Based Static` network binding only supports the field notifier. Getter or Setter methods are not supported.⌋

7.2.2.2.6 Serialization of Payload

In case of the static signal-service-translation always the signal-service-translation is responsible for the handling of the serialization.

[SWS_CM_80510] Ignoring not mapped elements

Upstream requirements: RS_CM_00004

⌈To allow migration the deserialization shall ignore signals which are not subject to `ServiceInstanceToSignalMapping`.⌋

7.2.2.2.7 De-Serialization of Payload

[SWS_CM_80511] Deserializing incomplete data belonging to a field

Upstream requirements: RS_CM_00004

⌈If less data than expected shall be de-serialized and the data to be de-serialized belong to a `Field`, the `initValue` shall be used if it is defined. Otherwise the data shall be completely discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

Based on [SWS_CM_12004]:

[SWS_CM_80514] Deserializing incomplete data on the proxy side belonging to a statically defined event and `eventReceptionDefaultValue` is defined

Upstream requirements: RS_CM_00004

⌈If less data than expected shall be deserialized on the proxy side and the data to be deserialized belongs to an Event and the `UserDefinedEventDeployment` has an `eventReceptionDefaultValue` defined, then the `eventReceptionDefaultValue` shall be used as a substitute for the missing data.⌋

Based on [SWS_CM_12005]:

[SWS_CM_80515] Deserializing incomplete data on the proxy side belonging to a statically defined event and `eventReceptionDefaultValue` is not defined

Upstream requirements: RS_CM_00004

⌈If less data than expected shall be deserialized on the proxy side and the data to be deserialized belongs to an Event and the `UserDefinedEventDeployment` has no `eventReceptionDefaultValue` defined, then the data shall be discarded and the incident shall be logged.⌋

7.2.2.3 Signal-Based IEEE1722 ACF Network binding

The Signal-Based IEEE1722 ACF network binding utilizes the IEEE1722 ACF stream according to [19] to transport CAN and LIN frames which in turn may contain signal-based ISignalIPdus.

On which EthernetCommunicationConnector the IEEE1722 ACF stream transported is defined by UserDefinedServiceInstanceToMachineMapping.communicationConnector (see also [TPS_MANI_03684]).

The criteria defined in [TPS_MANI_03682] are used to identify whether a specific IEEE1722 ACF stream may contain messages which are relevant for signal-service-translation. Which individual CAN and LIN frames are involved in a specific signal-service-translation is defined by [TPS_MANI_03683].

7.2.2.3.1 Service Discovery

The Signal-Based IEEE1722 ACF network binding does not support service discovery. Every involved IEEE1722 ACF stream is transported on a statically configured channel.

[SWS_CM_00057] Mapping of Offer Service in case of Signal-Based IEEE1722 ACF network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004, RS_CM_00204

⌈If instructed to *offer* a service instance which is mapped to a ProvidedUserDefinedServiceInstance of category SIGNALBASED_IEEE1722_ACF, then the Signal-Based IEEE1722 ACF network binding shall create / use a socket for the communication on the EthernetCommunicationConnector that the ProvidedUserDefinedServiceInstance is mapped to via the UserDefinedServiceInstanceToMachineMapping.communicationConnector. If a socket with identical credentials already exists, then this existing socket shall be used.⌋

[SWS_CM_00058] Mapping of Stop Offer Service in case of Signal-Based IEEE1722 ACF network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004, RS_CM_00204

⌈If instructed to *stop offering* a service instance which is mapped to a ProvidedUserDefinedServiceInstance of category SIGNALBASED_IEEE1722_ACF, then the Signal-Based IEEE1722 ACF network binding shall check:

- If this is the last communication which uses the respective socket, then this socket shall be closed.

- If there is still other communication using this socket, then the socket shall be kept open.

⌋

[SWS_CM_00059] Mapping of Find Service in case of `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004, RS_CM_00204

⌈If instructed to *find* a service instance which is mapped to a `RequiredUserDefinedServiceInstance` of `category` `SIGNALBASED_IEEE1722_ACF`, then the `Signal-Based IEEE1722 ACF` network binding shall immediately return a `ara::com::ServiceHandleContainer`.⌋

[SWS_CM_00060] Mapping of Subscribe Service in case of `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004, RS_CM_00204

⌈If instructed to *subscribe* to an event which is part of a `RequiredUserDefinedServiceInstance` of `category` `SIGNALBASED_IEEE1722_ACF`, then the `Signal-Based IEEE1722 ACF` network binding shall:

If there is not already a socket established: use the information from the `EthernetCommunicationConnector` and create the socket.

If there is already a socket established: use this socket.⌋

[SWS_CM_00061] Mapping of Unsubscribe Service in case of `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004, RS_CM_00204

⌈If instructed to *un-subscribe* from an event which is part of a `RequiredUserDefinedServiceInstance` of `category` `SIGNALBASED_IEEE1722_ACF`, then the `Signal-Based IEEE1722 ACF` network binding shall check:

- If this is the last communication which uses the respective socket, then this socket shall be closed.
- If there is still other communication using this socket, then the socket shall be kept open.

⌋

7.2.2.3.2 Accumulation of messages

An IEEE1722 ACF stream is capable of transporting several CAN and/or LIN frames in one message. For the reception this means that each received CAN and/or LIN frame needs to be processed individually by the Signal-Based IEEE1722 ACF network binding. The individual CAN and LIN frames are defined using the IEEE1722TpAcfBusPart class (IEEE1722TpAcfCanPart or IEEE1722TpAcfLinPart respectively).

[SWS_CM_00062] Data accumulation on a RequiredSomeipServiceInstance in case of Signal-Based IEEE1722 ACF network binding

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If a received Signal-Based IEEE1722 ACF message contains several CAN and/or LIN frames (IEEE1722TpAcfCanPart or IEEE1722TpAcfLinPart respectively), then each contained frame (IEEE1722TpAcfBusPart) shall be processed individually by the receiving Signal-Based IEEE1722 ACF network binding. Which individual CAN and LIN frames are involved in a specific signal-service-translation is defined by [TPS_MANI_03683].⌋

[SWS_CM_00063] Configuration of data accumulation on an IEEE1722 ACF stream

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If an IEEE1722TpAcfConnection has defined a value for collectionThreshold, collectionTimeout, and mixedBusTypeCollection, then the Signal-Based IEEE1722 ACF network binding shall implement accumulation of messages for that IEEE1722 ACF stream.⌋

The attribute IEEE1722TpAcfConnection.mixedBusTypeCollection defines whether the accumulation of messages shall respect the kind of IEEE1722TpAcfBusPart.

[SWS_CM_00067] Configuration of IEEE1722TpAcfConnection.mixedBusTypeCollection

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If IEEE1722TpAcfConnection.mixedBusTypeCollection is set to false, then each transmitted IEEE1722TpAcfConnection message shall have either IEEE1722TpAcfCanPart or IEEE1722TpAcfLinPart entries only.⌋

[SWS_CM_00066] Configuration of no data accumulation on an `IEEE1722 ACF stream`

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If an `IEEE1722TpAcfBusPart` is to be transmitted on an `IEEE1722TpAcfConnection` that does not qualify for accumulation of messages according to [SWS_CM_00063], then the `Signal-Based IEEE1722 ACF` network binding shall put the `IEEE1722TpAcfBusPart` into the buffer and immediately trigger the `IEEE1722TpAcfConnection` buffer for transmission.⌋

[SWS_CM_00064] Configuration of data accumulation on an `IEEE1722 ACF stream` in case of `collectionTrigger` equals `always`

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If an `IEEE1722TpAcfBusPart` is to be transmitted on an `IEEE1722TpAcfConnection` that qualifies for accumulation of messages according to [SWS_CM_00063] and the `IEEE1722TpAcfBusPart.collectionTrigger` is set to `PduCollectionTriggerEnum.always`, then

the `Signal-Based IEEE1722 ACF` network binding shall put the `IEEE1722TpAcfBusPart` into the buffer and immediately trigger the `IEEE1722TpAcfConnection` buffer for transmission.⌋

[SWS_CM_00065] Configuration of data accumulation on an `IEEE1722 ACF stream` in case of `collectionTrigger` equals `never`

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If an `IEEE1722TpAcfBusPart` is to be transmitted on an `IEEE1722TpAcfConnection` that qualifies for accumulation of messages according to [SWS_CM_00063] and the `IEEE1722TpAcfBusPart.collectionTrigger` is set to `PduCollectionTriggerEnum.never`, then

the `Signal-Based IEEE1722 ACF` network binding shall put the `IEEE1722TpAcfBusPart` into the buffer and trigger the `IEEE1722TpAcfConnection` buffer for transmission only if at least one of the following conditions applies:

- the `collectionTimeout` is reached for the `IEEE1722 ACF stream` message buffer.
- the buffer size defined by the attribute `collectionThreshold` is reached or exceeded.
- adding the `IEEE1722TpAcfBusPart` to the buffer would lead to an `IEEE1722 ACF stream` message larger than the maximum supported size (e.g. MTU size). In this case the actual buffer shall be triggered before handling the new `IEEE1722TpAcfBusPart`.

⌋

7.2.2.3.3 Support for Non-Time-Synchronous Control Format and Time-Synchronous Control Format

The Signal-Based IEEE1722 ACF network binding supports two kinds of control messages of IEEE1722 ACF streams according to [19]:

- Non-Time-Synchronous Control Format, where no presentation time is sent, and
- Time-Synchronous Control Format, where a presentation time is calculated and sent in the message.

If a IEEE1722 ACF stream message is triggered for transmission it depends on the existence of the attribute IEEE1722TpAcfConnection.acfMaxTransitTime which kind of Control Format is used.

[SWS_CM_00070] Transmission of Non-Time-Synchronous Control Format messages

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If an IEEE1722 ACF stream message is triggered for transmission and the attribute IEEE1722TpAcfConnection.acfMaxTransitTime is not defined, then the IEEE1722 ACF stream message shall be sent according to the "Non-Time-Synchronous Control Format" of [19].⌋

[SWS_CM_00071] Transmission of Time-Synchronous Control Format messages

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If an IEEE1722 ACF stream message is triggered for transmission and the attribute IEEE1722TpAcfConnection.acfMaxTransitTime is defined, then the IEEE1722 ACF stream message shall be sent according to the "Time-Synchronous Control Format" of [19].

The value used for the presentation time shall be calculated by taking the current time from the GlobalTimeDomain referenced by IEEE1722TpConnection.globalTimeDomain and adding the value IEEE1722TpAcfConnection.acfMaxTransitTime.⌋

[SWS_CM_00072] Reception of Non-Time-Synchronous Control Format messages

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If an `IEEE1722 ACF stream` message is received and the attribute `IEEE1722TpAcfConnection.acfMaxTransitTime` is not defined, then the `IEEE1722 ACF stream` message shall be processed immediately.⌋

[SWS_CM_00073] Reception of Time-Synchronous Control Format messages

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00006, RS_CM_00004

⌈If an `IEEE1722 ACF stream` message is received and the attribute `IEEE1722TpAcfConnection.acfMaxTransitTime` is defined, then the `IEEE1722 ACF stream` message presentation time shall be checked:

- if the current time of the `GlobalTimeDomain` referenced by `IEEE1722TpConnection.globalTimeDomain` is greater than the presentation time of the received `IEEE1722 ACF stream` message (the message was received too late), then the `IEEE1722 ACF stream` message shall be dropped.
- if the current time of the `GlobalTimeDomain` referenced by `IEEE1722TpConnection.globalTimeDomain` is smaller or equal than the presentation time of the received `IEEE1722 ACF stream` message, than the `IEEE1722 ACF stream` message shall be buffered until the presentation time is reached.

⌋

7.2.2.3.4 Handling Events

[SWS_CM_00076] Unconditional sending of an `IEEE1722 ACF stream` event message

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_CM_00006, RS_CM_00004

⌈The sending of an event message mapped to an `IEEE1722 ACF stream` shall be requested by invoking the `Send` method of the respective `Event` class (see [SWS_CM_00162] and [SWS_CM_90437]).⌋

[SWS_CM_00077] IEEE1722 ACF Protocol used for sending of an `IEEE1722 ACF stream` event message

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00010, RS_CM_00006, RS_CM_00004

⌈The event message shall be transmitted using the `IEEE1722 ACF stream` according to [19].⌋

[SWS_CM_00078] Destination of an `IEEE1722 ACF stream` event message

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042, RS_CM_00006, RS_CM_00004

⌈The event message shall be transmitted using the `IEEE1722 ACF stream` according to [19] on the `EthernetCommunicationConnector` defined by the `UserDefinedServiceInstanceToMachineMapping.communicationConnector` and to the MAC address:

- if `IEEE1722TpConnection.destinationMacAddress` is defined, then to `destinationMacAddress`
- if `IEEE1722TpConnection.destinationMacAddress` is not defined, then to `IEEE1722TpConnection.macAddressStreamId`.

⌋

[SWS_CM_00079] Checks for a received signal-based serialized `IEEE1722 ACF stream` event message

Status: DRAFT
Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00019, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_SOMEIP_00008, RS_SOMEIP_00014, RS_CM_00006, RS_CM_00004

⌈If an `ISignalIPdu` qualifies for reception according to [TPS_MANI_03683] and any of the contained `ISignal`s in that `ISignalIPdu` (via the `SignalBasedEventElementToISignalTriggeringMapping.iSignalTriggering`) has a `SignalBasedEventElementToISignalTriggeringMapping`, then these `ISignal`s shall be processed for reception.

Otherwise the received `IEEE1722 ACF stream` signal-based event message shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋

[SWS_CM_00080] Deserializing the `IEEE1722 ACF stream` signal-based serialized payload

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004

⌈The Payload of the `IEEE1722 ACF stream` signal-based serialized event message (i.e. the serialized `VariableDataPrototype` composed by the `ServiceInterface` in role `event` mapped via the `ISignalTriggering` by the `SignalBasedEventElementToISignalTriggeringMapping`) shall be deserialized according to the signal-service-translation serialization rules defined in TPS-ManifestSpecification [5].⌋

7.2.2.3.5 Handling Method Calls

[SWS_CM_00068] No method support for `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004, RS_CM_00204

⌈The `Signal-Based IEEE1722 ACF` network binding does not support methods.⌋

7.2.2.3.6 Handling Fields

[SWS_CM_00069] Only field notifier support for `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004, RS_CM_00204

⌈The `Signal-Based IEEE1722 ACF` network binding only supports the field notifier. Getter or Setter methods are not supported.⌋

7.2.2.3.7 Serialization of Payload

[SWS_CM_00074] Ignoring not mapped elements in case of `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004

⌈To allow migration the deserialization shall ignore signals which are not subject to `ServiceInstanceToSignalMapping`.⌋

7.2.2.3.8 De-Serialization of Payload

[SWS_CM_00075] Deserializing incomplete data belonging to a field in case of `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00004

⌈If less data than expected shall be deserialized and the data to be deserialized belong to a `Field`, the `initValue` shall be used if it is defined. Otherwise the data shall be completely discarded and the incident shall be logged.⌋

[SWS_CM_00081] Deserializing incomplete data on the proxy side belonging to an event and `eventReceptionDefaultValue` is defined in case of `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00204, RS_CM_00202

⌈If less data than expected shall be deserialized on the proxy side and the data to be deserialized belongs to an Event and the `eventReceptionDefaultValue` is defined, then the `eventReceptionDefaultValue` shall be used as a substitute for the missing data.⌋

[SWS_CM_00082] Deserializing incomplete data on the proxy side belonging to an event and `eventReceptionDefaultValue` is not defined in case of `Signal-Based IEEE1722 ACF` network binding

Status: DRAFT
Upstream requirements: RS_CM_00006, RS_CM_00204, RS_CM_00202

⌈If less data than expected shall be deserialized on the proxy side and the data to be deserialized belongs to an Event and the `eventReceptionDefaultValue` is not defined, then the data shall be discarded and the incident shall be logged.⌋

7.2.2.4 Execution context of message reception actions

The section 7.2.1.4 is fully applicable to the signal-based network binding.

7.2.3 DDS Network binding

[SWS_CM_11000] DDS Compliance

Upstream requirements: RS_CM_00204, FO_RS_Dds_00001

⌈The DDS network binding shall comply with the DDS Minimum Profile defined in [20], the DDS Wire Interoperability protocol (RTPS) defined in [21], and the DDS-XTYPES Minimal Programming Interface and Network Interoperability Profiles defined in [22].⌋

[SWS_CM_90500] Choice of Service Instance discovery protocol

Upstream requirements: RS_CM_00101, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_-00008

⌈`DdsProvidedServiceInstance`s and `DdsRequiredServiceInstance`s provide a `discoveryType` attribute permitting the choice between two distinct discovery protocols. For a Service Interface Skeleton to be discoverable by a Service Interface Proxy, both shall be configured with the same `discoveryType` value.⌋

The `DomainParticipantUserDataQos` setting provides a discovery protocol that leverages the USER_DATA QoS policy of DDS Domain Participants, assigning a purpose-specific format string to it as described in 7.2.3.1 below. This approach is fast and nimble, since no additional DDS Entities beyond Domain Participants need to be created to exercise discovery of Service Instances.

The `Topic` setting provides, as described in section 7.2.3.2 below, a discovery protocol that employs a purpose-specific Topic of a well-defined type to distribute Service Instance announcements in a publish-subscribe, instance-based fashion. This protocol, although more resource-demanding (DDS entities down to a single DataWriter need to be created for Skeletons, same for a DataReader in Proxies), enhances interoperability and enables advanced DDS features such as persistence, routing and durability.

[SWS_CM_90501] Topic naming for Domain Participant USER_DATA QoS - based Service Instances

Upstream requirements: RS_CM_00201, RS_CM_00211, RS_CM_00216, FO_RS_Dds_00001, FO_RS_Dds_00005

⌈When `DomainParticipantUserDataQos` is set in the `discoveryType` attribute for a specific `DdsProvidedServiceInstance` or `DdsRequiredServiceInstance`, the de-facto Topic naming scheme for events, triggers, methods and fields is the one described for `SERVICE_INSTANCE_RESOURCE_PARTITION`.⌋

7.2.3.1 Service Discovery via Domain Participant USER_DATA QoS policy

[SWS_CM_11001] Mapping of OfferService method

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_00005

⌈The binding implementation shall configure the USER_DATA QoS Policy of the DDS DomainParticipant associated with the Service Instance to propagate Service IDs, Instance IDs, and `ServiceInterface` contract versions, as described by [FO_PRS_DDSSD_00102], where:

- Service Interface Identifier is derived from `DdsServiceInterfaceDeployment` `serviceInterfaceId`.
- Service Instance Identifier is derived from `DdsProvidedServiceInstance` `serviceInstanceId`.
- Service Interface contract major version is derived from `ServiceInterface` `majorVersion`.
- Service Interface contract minor version is derived from `ServiceInterface` `minorVersion`.

⌋

[SWS_CM_11002] Assigning a DDS DomainParticipant to a Service Instance

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_00005

⌈The DDS Binding shall assign a DDS DomainParticipant to every Service Instance as described in [FO_PRS_DDSSD_00101], where:

- The Domain ID of the DomainParticipant shall be derived from the Manifest, where the `DdsProvidedServiceInstance` element defines the `domainId`.
- The QoS Profile of the DomainParticipant shall be derived from the Manifest, where the `DdsProvidedServiceInstance` element defines the `qosProfile`.

Before creating a new DomainParticipant, the DDS binding shall first look for existing DomainParticipants in the current process that match the configuration criteria specified above[3]. If the search is successful, the binding shall assign the DomainParticipant found to the Service[4]; otherwise, the binding shall create a new DomainParticipant according to the desired configuration and assign it to the Service.

Once the DomainParticipant is available to the Service Instance, the binding implementation shall create a DDS Publisher and a DDS Subscriber to enclose all DataWriters

[3]The DDS APIs that provide the ability to find existing DomainParticipants search in the scope of the address space of the current process—only local DomainParticipants may be reused.

[4]The rules specified in this binding ensure the creation of only one DomainParticipant for a given Domain and set of QoS settings (`qosProfile`).

and DataReaders associated with the Instance. The Partition QoS of both the DDS Publisher and DDS Subscriber shall contain the following partition name:

`"ara.com://services/<svcId>_<svcInId>"`

Where:

<svcId> is the Service Id derived from the Manifest, where the DdsServiceInterfaceDeployment element defines the serviceInterfaceId.

<svcInId> is the Instance Id derived from the Manifest, where the DdsProvidedServiceInstance element defines the serviceInstanceId.

Publisher and Subscriber objects may be reused across events and other resources provided by the Service Instance; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.

⌋

[SWS_CM_11003] Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00015

⌈The DDS binding shall assign a DDS Topic to every event in the ServiceInterface according to the mapping rules specified in [SWS_CM_11015]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create a new DDS Topic to represent the event as defined in [SWS_CM_11015].

Once all DDS Topics representing the events in the ServiceInterface are ready for use, the DomainParticipant assigned to the Service Instance shall create one DDS DataWriter of the equivalent Topic per event using the DDS Publisher created in [SWS_CM_11002]. The DataWriter shall be configured according to the qosProfile specified in the associated DdsEventQosProps.

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

[SWS_CM_10550] Assigning a DDS Topic and a DDS DataWriter to every Trigger in the ServiceInterface

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00015

⌈The DDS binding shall assign a DDS Topic to every trigger in the ServiceInterface according to the mapping rules specified in [SWS_CM_10524]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service

Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create a new DDS Topic to represent the `trigger` as defined in [SWS_CM_10524].

Once all DDS Topics representing the `trigger`s in the `ServiceInterface` are ready for use, the DomainParticipant assigned to the Service Instance shall create one DDS DataWriter of the equivalent Topic per `trigger` using the DDS Publisher created in [SWS_CM_11002]. The DataWriter shall be configured according to the `qosProfile` specified in the associated `DdsEventQosProps` that in turn refers via `DdsEventDeployment` to the `trigger`s.

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

[SWS_CM_11029] Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Methods in the ServiceInterface

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_-00015

⌈The DDS binding shall instantiate a DDS Service [23] to handle requests to all the `method`s in the `ServiceInterface`.

In practice, this implies assigning a DDS Request Topic and a DDS Reply Topic to the DDS Service that handles those method calls according to the mapping rules specified in [SWS_CM_11100]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create new DDS Request and Reply Topics to represent the DDS Service as specified in [SWS_CM_11100].

Once the corresponding DDS Request and Reply Topics are ready for use, the DomainParticipant assigned to the Service Instance shall create:

- [SWS_CM_11106] A DDS DataReader of the DDS Request Topic to handle requests using the DDS Subscriber created in [SWS_CM_11002].
- [SWS_CM_11107] A DDS DataWriter of the DDS Reply Topic to handle replies using the DDS Publisher created in [SWS_CM_11002].

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

The handling of method calls with DDS is specified in 7.2.3.5.

[SWS_CM_11030] Assigning a DDS Topic and a DDS DataWriter to every Field in the ServiceInterface with its hasNotifier attribute equal to true

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_-00008, FO_RS_Dds_00015

⌈The DDS binding shall assign a DDS Topic to every `field` in the `ServiceInterface` with its `hasNotifier` attribute set to `true` according to the mapping rules specified in [SWS_CM_11130]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create a new DDS Topic to represent the `field` as defined in [SWS_CM_11130].

Once all DDS Topics representing the `field`s in the `ServiceInterface` are ready for use, the DomainParticipant assigned to the Service Instance shall create one DDS DataWriter of the equivalent Topic per `field` with the `hasNotifier` attribute set to `true` using the DDS Publisher created in [SWS_CM_11002]. The DataWriter shall be configured according to the `qosProfile` specified in the associated `DdsField-QosProps`.

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

[SWS_CM_11031] Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Field Getters/Setters in the ServiceInterface

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_-00008, FO_RS_Dds_00015, FO_RS_Dds_00016

⌈The DDS binding shall instantiate a DDS Service [23] to handle get/set requests to all the `field`s in the `ServiceInterface` with `hasGetter` and/or `hasSetter` set to `true`.

In practice, this implies assigning a DDS Request Topic and a DDS Reply Topic to the DDS Service according to the mapping rules specified in [SWS_CM_11144]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create new DDS Request and Reply Topics to represent the DDS Service as specified in [SWS_CM_11144].

Once the corresponding DDS Request and Reply Topics are ready for use, the DomainParticipant assigned to the Service Instance shall create:

- [SWS_CM_11149] A DDS DataReader of the DDS Request Topic to handle requests using the DDS Subscriber created in [SWS_CM_11002].
- [SWS_CM_11150] A DDS DataWriter of the DDS Reply Topic to handle replies using the DDS Publisher created in [SWS_CM_11002].

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

The handling of fields with DDS is specified in section 7.2.3.6.

[SWS_CM_09004] Adding Service IDs, Service Instance IDs, and ServiceInterface Contract Versions to the DDS DomainParticipant's USER_DATA QoS Policy

Upstream requirements: RS_CM_00200, RS_CM_00101, RS_CM_00500, RS_CM_00501, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008

⌈The binding implementation shall configure the USER_DATA QoS Policy of the DDS DomainParticipant associated with the Service Instance to propagate Service IDs, Instance IDs, and `ServiceInterface` contract versions, as described by [FO_PRS_DDSSD_00102], where:

- Service Interface Identifier is derived from `DdsServiceInterfaceDeployment` `serviceInterfaceId`.
- Service Instance Identifier is derived from `DdsProvidedServiceInstance` `serviceInstanceId`.
- Service Interface contract major version is derived from `ServiceInterface` `majorVersion`.
- Service Interface contract minor version is derived from `ServiceInterface` `minorVersion`.

⌋

[SWS_CM_11005] Mapping of StopOfferService method

Upstream requirements: RS_CM_00105, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008, FO_RS_Dds_00015, FO_RS_Dds_00016

⌈When instructed to stop offering a Service, the DDS Binding shall perform the following operations:

- It shall remove all DDS DataWriters associated with `event`s in the `ServiceInterface` created in previous calls to the `OfferService()` method.
- It shall remove all DDS DataWriters associated with `trigger`s in the `ServiceInterface` created in previous calls to the `OfferService()` method.
- It shall remove all DDS DataWriters and DataReaders associated with the `ClientServerOperation`s defined in the role `method` created in previous calls to the `OfferService()` method.
- It shall remove all DDS DataWriters associated with `field`s in the `ServiceInterface` with their `hasNotifier` attribute set to `true` created in previous calls to the `OfferService()` method.

- It shall remove all DDS DataWriters and DataReaders associated with the `fields` in the `ServiceInterface` with `hasGetter` and/or `hasSetter` attributes set to `true` created in previous calls to the `OfferService()` method.
- It shall follow steps described in [FO_PRS_DDSSD_00103].

⌋

[SWS_CM_11006] Mapping of FindService method

Upstream requirements: RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_-00005, FO_RS_Dds_00008

⌈When instructed to find remote Services, the DDS Binding shall perform the following operations:

- [SWS_CM_11007] It shall look for an existing DDS DomainParticipant capable of finding remote Services Instances. If such DomainParticipant does not exist, the DDS binding shall create a new one as specified in [SWS_CM_11008].
- [SWS_CM_11009] It shall iterate, as described in [FO_PRS_DDSSD_00106], over the list of discovered remote DomainParticipants and look for those associated with Service Instances that: (1) match the filter criteria specified in the `FindService()` call, (2) have a compatible `ServiceInterface` contract version, and (3) have a `ServiceInterface` contract version that is not part of a `DdsRequiredServiceInstance.blocklistedVersion`.
- It shall return a `hierarchicalnamespacelistlowerproxy::proxy::serviceinterfacenameuppercamelProxy::HandleType` object for every Service Instance that: (1) matches the filter criteria, (2) has a compatible `ServiceInterface` contract version, and (3) has a `ServiceInterface` contract version that is not part of a `DdsRequiredServiceInstance.blocklistedVersion`. The Handle object shall contain a reference to both the DomainParticipant that was used in the discovery phase and the DDS Publisher and Subscriber created to match the partition of the remote service instance (see [SWS_CM_11009]), so that they can be used to create the appropriate DataWriters and DataReaders to handle remote communication.

⌋

[SWS_CM_11007] Finding a DDS DomainParticipant suitable for performing client-side operations

Upstream requirements: RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_-00005, FO_RS_Dds_00008

⌈The DDS binding shall provide client-side proxies with a DDS DomainParticipant as described by [FO_PRS_DDSSD_00105], where:

- The Domain ID of the DomainParticipant is derived from `DdsRequiredServiceInstance domainId`.

- The QoS Profile of the DomainParticipant is derived from `DdsRequiredServiceInstance qosProfile`.

⌋

[SWS_CM_11008] Creating a DDS DomainParticipant suitable for performing client-side operations

Upstream requirements: RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008

⌈To create a DomainParticipant capable of discovering and communicating with remote DDS DomainParticipants assigned to Service Instances, the binding implementation shall follow [FO_PRS_DDSSD_00105], using the configuration parameters in the TPS_ManifestSpecification described in [SWS_CM_11007].⌋

[SWS_CM_11009] Discovering remote Service Instances through DDS DomainParticipants

Upstream requirements: RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008, RS_CM_00501, RS_CM_00701

⌈To retrieve the list of discovered Service Instances, the DDS binding shall operate as described by [FO_PRS_DDSSD_00106], where:

- Service Interface Identifier is derived from `DdsServiceInterfaceDeployment serviceInterfaceId`.
- Service Instance Identifier is derived from `DdsProvidedServiceInstance serviceInstanceId`.
- Service Interface contract major version is derived from `ServiceInterface majorVersion`.
- Service Interface contract minor version is derived from `ServiceInterface minorVersion`.

And:

- If `requiredServiceInstanceId` is set to `ALL`, the binding shall return a new handle for each Service Instance found.
- Otherwise, the binding shall return a new handle only for Service Instances found that match the defined Service Instance Identifier.
- The `ServiceInterface` contract version of the discovered service instance is compatible with the `serviceInterfaceDeployment` version of the `DdsRequiredServiceInstance` according to [RS_CM_00501].
- The `ServiceInterface` contract version is not part of any `DdsRequiredServiceInstance blocklistedVersion`, according to [RS_CM_00701].

⌋

[SWS_CM_11010] Mapping of StartFindService method

Upstream requirements: RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_-00005, FO_RS_Dds_00008

⌈When instructed to start a continuous service search, the DDS Binding shall perform the following operations:

- [SWS_CM_11007] It shall look for an existing DDS DomainParticipant capable of finding remote Service Instances. If such DomainParticipant does not exist, the DDS binding shall create it as specified in [SWS_CM_11008].
- [SWS_CM_11011] It shall define a DDS BuiltinParticipantListener capable of calling the given `ara::com::FindServiceHandler` upon the occurrence of any of the following events:
  1. A remote DomainParticipant assigned to a matching Service is discovered.
  2. A remote DomainParticipant assigned to a matching Service does not contain the service anymore (i.e., any time a remote DomainParticipant stopped offering a matching Service by removing it from its USER_DATA QoS).
  3. A remote DomainParticipant assigned to a matching Service ceases to exist (i.e., the instance state is either `NOT_ALIVE_DISPOSED` or `NOT_ALIVE_NO_WRITERS`).
- [SWS_CM_11012] It shall bind the defined BuiltinParticipantListener to the DomainParticipant.

⌋

[SWS_CM_11011] Defining a DDS BuiltinParticipantListener

Upstream requirements: RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_-00005, FO_RS_Dds_00008

⌈The DDS Binding implementation shall define a Domain Participant Built-in Topic Listener class as described by [FO_PRS_DDSSD_00108].⌋

[SWS_CM_11012] Binding a BuiltinParticipantListener to a DDS DomainParticipant

Upstream requirements: RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_-00005, FO_RS_Dds_00008

⌈The DDS Binding implementation shall bind a Domain Participant Built-in Topic Listener as described by [FO_PRS_DDSSD_00109].⌋

[SWS_CM_11013] Mapping of `StopFindService()` method

Upstream requirements: RS_CM_00200, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008

⌈When instructed to stop a continuous service search initiated by a previous call to `StartFindService()`, the DDS Binding implementation shall proceed as described by [FO_PRS_DDSSD_00111].⌋

[SWS_CM_11014] Unbinding a BuiltinParticipantListener from a DDS DomainParticipant

Upstream requirements: RS_CM_00200, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008

⌈When instructed to unbind a Domain Participant Built-in Topic Listener from a DDS DomainParticipant, the DDS binding implementation shall proceed as described by [FO_PRS_DDSSD_00111].⌋

7.2.3.2 Service Discovery via Topic

[SWS_CM_90502] Mapping of OfferService method

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to offer a Service, the DDS Binding shall perform the following operations:

- [SWS_CM_90503] It shall assign a DDS DomainParticipant to the Service Instance.
- [SWS_CM_90504] It shall assign a DDS Topic and a DDS DataWriter to every `VariableDataPrototype` defined in the `ServiceInterface` in the role `event`.
- [SWS_CM_90505] It shall assign a DDS Request Topic and a DDS Reply Topic, and create their corresponding DDS DataWriter and DataReader, to provide access to all `ClientServerOperation`s defined in the `ServiceInterface` the role `method`.
- [SWS_CM_90506] It shall assign a DDS Topic and a DDS DataWriter to every `Field` defined in the `ServiceInterface` in the role `field` with its `hasNotifier` attribute set to `true`.
- [SWS_CM_90507] It shall assign a DDS Request Topic and a DDS Reply Topic, and create their corresponding DDS DataWriter and DDS DataReader, to provide access to all the `Field`s defined in the `ServiceInterface` in the role `field` with `hasGetter` and/or `hasSetter` attributes set to `true` via getter/setter invocation.

- [SWS_CM_90508] It shall advertise the Service Interface ID, Service Instance ID, and `ServiceInterface` contract version via the `ara.com://services/discovery` DDS topic

⌋

[SWS_CM_90503] Assigning a DDS DomainParticipant to a Service Instance

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015, FO_RS_Dds_00016

⌈The DDS Binding shall assign a DDS DomainParticipant to every Service Instance as described in [FO_PRS_DDSSD_00201], where:

- The Domain ID of the DomainParticipant shall be derived from the Manifest, where the `DdsProvidedServiceInstance` element defines the `domainId`.
- The QoS Profile of the DomainParticipant shall be derived from the Manifest, where the `DdsProvidedServiceInstance` element defines the `qosProfile`.

⌋

[SWS_CM_90504] Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈The DDS binding shall assign a DDS Topic to every `event` in the `ServiceInterface` according to the mapping rules specified in [SWS_CM_11015]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create a new DDS Topic to represent the `event` as defined in [SWS_CM_11015].

Once all DDS Topics representing the `event`s in the `ServiceInterface` are ready for use, the DomainParticipant assigned to the Service Instance shall create one DDS DataWriter of the equivalent Topic per `event` using the DDS Publisher created in [SWS_CM_90503]. The DataWriter shall be configured according to the `qosProfile` specified in the associated `DdsEventQosProps`.

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

[SWS_CM_90505] Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Methods in the ServiceInterface

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_-00008, FO_RS_Dds_00015, FO_RS_Dds_00016

⌈The DDS binding shall instantiate a DDS Service [23] to handle requests to all the `methods` in the `ServiceInterface`.

In practice, this implies assigning a DDS Request Topic and a DDS Reply Topic to the DDS Service that handles those method calls according to the mapping rules specified in [SWS_CM_11100]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create new DDS Request and Reply Topics to represent the DDS Service as specified in [SWS_CM_11100].

Once the corresponding DDS Request and Reply Topics are ready for use, the DomainParticipant assigned to the Service Instance shall create:

- [SWS_CM_11106] A DDS DataReader of the DDS Request Topic to handle requests using the DDS Subscriber created in [SWS_CM_90503].
- [SWS_CM_11107] A DDS DataWriter of the DDS Reply Topic to handle replies using the DDS Publisher created in [SWS_CM_90503].

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

The handling of method calls with DDS is specified in 7.2.3.5.

[SWS_CM_90506] Assigning a DDS Topic and a DDS DataWriter to every Field in the ServiceInterface with its hasNotifier attribute equal to true

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_-00005, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈The DDS binding shall assign a DDS Topic to every `field` in the `ServiceInterface` with its `hasNotifier` attribute set to `true` according to the mapping rules specified in [SWS_CM_11130]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create a new DDS Topic to represent the `field` as defined in [SWS_CM_11130].

Once all DDS Topics representing the `fields` in the `ServiceInterface` are ready for use, the DomainParticipant assigned to the Service Instance shall create one DDS DataWriter of the equivalent Topic per `field` with the `hasNotifier` attribute set to `true` using the DDS Publisher created in [SWS_CM_90503]. The DataWriter shall

be configured according to the qosProfile specified in the associated DdsFieldQosProps.

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

[SWS_CM_90507] Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Field Getters/Setters in the ServiceInterface

Upstream requirements: RS_CM_00200, RS_CM_00101, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015, FO_RS_Dds_00016

⌈The DDS binding shall instantiate a DDS Service [23] to handle get/set requests to all the fields in the ServiceInterface with hasGetter and/or hasSetter set to true.

In practice, this implies assigning a DDS Request Topic and a DDS Reply Topic to the DDS Service according to the mapping rules specified in [SWS_CM_11144]. Since these DDS Topics may already be available in the DomainParticipant assigned to the Service Instance (e.g., because a different Service Instance assigned to the same DomainParticipant may have created them), the service shall first look for existing Topics in the DomainParticipant matching the required criteria. If the search is unsuccessful, the DomainParticipant shall create new DDS Request and Reply Topics to represent the DDS Service as specified in [SWS_CM_11144].

Once the corresponding DDS Request and Reply Topics are ready for use, the DomainParticipant assigned to the Service Instance shall create:

- [SWS_CM_11149] A DDS DataReader of the DDS Request Topic to handle requests using the DDS Subscriber created in [SWS_CM_90503].
- [SWS_CM_11150] A DDS DataWriter of the DDS Reply Topic to handle replies using the DDS Publisher created in [SWS_CM_90503].

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋

The handling of fields with DDS is specified in section 7.2.3.6.

[SWS_CM_90508] Advertising Service IDs, Service Instance IDs, and ServiceInterface Contract Versions over the `ara.com://services/discovery` topic

Upstream requirements: RS_CM_00200, RS_CM_00101, RS_CM_00500, RS_CM_00501, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00007, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈The binding implementation shall configure DDS Topic, Publisher and DataWriter objects, as described by [FO_PRS_DDSSD_00202], where:

- Service Interface Identifier is derived from DdsServiceInterfaceDeployment serviceInterfaceId.

- Service Instance Identifier is derived from `DdsProvidedServiceInstance serviceInstanceId`.
- Service Interface contract major version is derived from `ServiceInterface majorVersion`.
- Service Interface contract minor version is derived from `ServiceInterface minorVersion`.
- Identifier Type is derived from `DdsProvidedServiceInstance resourceIdentifierType`, and defines the protocol used by consumers of the Service Instance to bind themselves with it. This choice will determine topic naming, usage of partitions and the relevance of in-band instance identifers in the following specification items: [SWS_CM_11015], [SWS_CM_11100], [SWS_CM_11130], [SWS_CM_11144] and [SWS_CM_10524].

⌋

[SWS_CM_90509] Mapping of StopOfferService method

Upstream requirements: RS_CM_00105, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to stop offering a Service, the DDS Binding shall perform the following operations:

- Dispose the discovery instance according to [FO_PRS_DDSSD_00203].
- It shall remove all DDS DataWriters associated with `event`s in the `ServiceInterface` created in previous calls to the `OfferService()` method.
- It shall remove all DDS DataWriters and DataReaders associated with the `ClientServerOperation`s defined in the role `method` created in previous calls to the `OfferService()` method.
- It shall remove all DDS DataWriters associated with `field`s in the `ServiceInterface` with their `hasNotifier` attribute set to `true` created in previous calls to the `OfferService()` method.
- It shall remove all DDS DataWriters and DataReaders associated with the `field`s in the `ServiceInterface` with `hasGetter` and/or `hasSetter` attributes set to `true` created in previous calls to the `OfferService()` method.

⌋

[SWS_CM_90510] Mapping of FindService method

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to find remote Services, the DDS Binding shall perform the following operations:

- [SWS_CM_90511] It shall look for an existing DDS DomainParticipant capable of finding remote Services Instances. If such DomainParticipant does not exist, the DDS binding shall create a new one as specified in [SWS_CM_90512].
- [SWS_CM_90513] It shall create an manipulate a DataReader as decribed in [FO_PRS_DDSSD_00206], looking into all samples received for those associated with Service Instances that: (1) match the filter criteria specified in the `FindService(InstanceIdentifier)` / `FindService(InstanceSpecifier)` call, (2) have a compatible `ServiceInterface` contract version, and (3) have a `ServiceInterface` contract version that is not part of a `DdsRequiredServiceInstance.blocklistedVersion`.
- It shall return a `hierarchicalnamespacelistlowerproxy::proxy::serviceinterfacenameuppercamelProxy::HandleType` object for every Service Instance that: (1) matches the filter criteria, (2) has a compatible `ServiceInterface` contract version, and (3) has a `ServiceInterface` contract version that is not part of a `DdsRequiredServiceInstance.blocklistedVersion`.

⌋

[SWS_CM_90511] Finding a DDS DomainParticipant suitable for performing client-side operations

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008

⌈The DDS binding shall provide client-side proxies with a DDS DomainParticipant as described by [FO_PRS_DDSSD_00205], where:

- The Domain ID of the DomainParticipant is derived from `DdsRequiredServiceInstance domainId`.
- The QoS Profile of the DomainParticipant is derived from `DdsRequiredServiceInstance qosProfile`.

⌋

[SWS_CM_90512] Creating a DDS DomainParticipant suitable for performing client-side operations

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈To create a DomainParticipant capable of discovering and communicating with remote DDS DomainParticipants assigned to Service Instances, the binding implementation shall follow [FO_PRS_DDSSD_00205], using the configuration parameters in the TPS_ManifestSpecification described in [SWS_CM_90511].⌋

[SWS_CM_90513] Discovering remote Service Instances through the ara.com://services/discovery topic

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈To retrieve the list of discovered Service Instances, the DDS binding shall operate as described by [FO_PRS_DDSSD_00206], where:

- Service Interface Identifier is derived from `DdsServiceInterfaceDeployment serviceInterfaceId`.
- Service Instance Identifier is derived from `DdsProvidedServiceInstance serviceInstanceId`.
- Service Interface contract major version is derived from `ServiceInterface majorVersion`.
- Service Interface contract minor version is derived from `ServiceInterface minorVersion`.

And:

- If `requiredServiceInstanceId` is set to `ALL`, the binding shall return a new handle for each Service Instance found.
- Otherwise, the binding shall return a new handle only for Service Instances found that match the defined Service Instance Identifier.
- The `ServiceInterface` contract version of the discovered service instance is compatible with the `serviceInterfaceDeployment` version of the `DdsRequiredServiceInstance` according to [RS_CM_00501].
- The `ServiceInterface` contract version is not part of any `DdsRequiredServiceInstance blocklistedVersion`, according to [RS_CM_00701].

⌋

[SWS_CM_90514] Mapping of StartFindService method

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to start a continuous service search, the DDS Binding shall perform the following operations:

- [SWS_CM_90511] It shall look for an existing DDS DomainParticipant capable of finding remote Service Instances. If such DomainParticipant does not exist, the DDS binding shall create it as specified in [SWS_CM_90512].
- It shall continuously monitor arrival of `ServiceAnnouncementMessage` samples through the `ara.com://services/discovery` topic, as described in [FO_PRS_DDSSD_00205] and [FO_PRS_DDSSD_00206] calling `ara::com::FindServiceHandler` whenever a matching Service Instance is discovered.

⌋

[SWS_CM_90515] Mapping of `StopFindService()` method

Upstream requirements: RS_CM_00204, RS_CM_00200, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to stop a continuous service search initiated by a previous call to `StartFindService()`, the DDS binding implementation shall proceed as described by [FO_PRS_DDSSD_00208].⌋

7.2.3.3 Handling Events

[SWS_CM_11015] Mapping Events to DDS Topics

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00007, FO_RS_Dds_00008

⌈The DDS binding shall map every `VariableDataPrototype` defined in the `ServiceInterface` in the role `event` to a DDS Topic, as described in [FO_PRS_DDS_00100].⌋

[SWS_CM_11016] DDS Topic data type definition

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00007

⌈The data type of a DDS Topic representing an Event shall be constructed, as described in [FO_PRS_DDS_00101].⌋

The DDS serialization rules are defined in section 7.2.3.7.

[SWS_CM_11017] Mapping of Send method

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00015

⌈When instructed to send an event message, the DDS Binding shall construct and send a new sample of the equivalent DDS Topic data type as described in [FO_PRS_DDS_00102].⌋

The DDS serialization rules are defined in section 7.2.3.7.

[SWS_CM_11018] Mapping of Subscribe method

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00016

⌈When instructed to subscribe to an event, the DDS binding shall create a DDS DataReader as described in [FO_PRS_DDS_00103].⌋

[SWS_CM_11019] Creating a DDS DataReader for event subscription

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00016, FO_RS_Dds_00005, FO_RS_Dds_00008

⌈The DDS binding shall configure a DDS DataReader for the Topic associated with the `event` (see [SWS_CM_11015]), as described in [FO_PRS_DDS_00104].⌋

[SWS_CM_11020] Defining a DDS DataReaderListener

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈The DDS Binding implementation shall define a `DataReaderListener` object capable of handling notifications when a new sample is received and/or when the matched status of the subscription changes for the Topic data type specified in [SWS_CM_11016], as described in [FO_PRS_DDS_00105].⌋

[SWS_CM_11021] Mapping of Unsubscribe method

Upstream requirements: RS_CM_00104, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unsubscribe from a service event, the DDS binding shall delete the DataReader associated with the `event`, as described in [FO_PRS_DDS_00106].⌋

[SWS_CM_11022] Mapping of `GetSubscriptionState()` method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to provide the subscription state, the DDS binding shall query the DataReader associated to the service event as described in [FO_PRS_DDS_00107].⌋

[SWS_CM_11023] Mapping of GetNewSamples method

Upstream requirements: RS_CM_00202, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to get new samples, the DDS binding shall perform a `take()` operation on the DataReader, as described in [FO_PRS_DDS_00108], with the following `GetNewSamples`-specific behaviour:

- If a `maxNumberOfSamples` is specified, the binding implementation shall invoke `take()` with `max_samples = maxNumberOfSamples`.
- Else, if no `maxNumberOfSamples` is specified (i.e., if `maxNumberOfSamples` is equal to the default value `std::numeric_limits<std::size_t>::max()`), the binding implementation shall invoke `take()` without specifying a `max_samples` limit.

After calling `take()`, the binding implementation shall invoke the `Callable f` for every valid sample taken from the DataReader's cache (i.e., every sample with `SampleInfo.valid_data` equal to `true`), providing `f` with a reference to the corresponding sample.

⌋

[SWS_CM_11024] Mapping of GetFreeSampleCount method

Upstream requirements: RS_CM_00202, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to provide the number of free sample slots, the binding implementation shall return the number free sample slots as described in [FO_PRS_DDS_00109].⌋

[SWS_CM_11025] Mapping of SetReceiveHandler method

Upstream requirements: RS_CM_00203, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to register an `EventReceiveHandler`, the binding implementation shall link the provided handler to the service event's DataReader Listener as described in [FO_PRS_DDS_00110].⌋

[SWS_CM_11026] Mapping of `UnsetReceiveHandler()` method

Upstream requirements: RS_CM_00203, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unregister an `EventReceiveHandler`, the binding implementation shall unlink the current handler, if any, as described in [FO_PRS_DDS_00111].⌋

[SWS_CM_11027] Mapping of SetSubscriptionStateHandler method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to register a `SubscriptionStateChangeHandler`, the binding implementation shall link the provided handler to the service event's DataReader Listener as described in [FO_PRS_DDS_00112].⌋

[SWS_CM_11028] Mapping of UnsetSubscriptionStateHandler method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unregister a `SubscriptionStateChangeHandler`, the binding implementation shall unlink the current handler, if any, as described in [FO_PRS_DDS_00113].⌋

7.2.3.4 Handling Triggers

[SWS_CM_10524] Mapping Triggers to DDS Topics

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00007, FO_RS_Dds_00008

⌈The DDS binding shall map every `Trigger` defined in the `ServiceInterface` in the role `trigger` to a DDS Topic, as described in [FO_PRS_DDS_00200].⌋

[SWS_CM_10525] DDS Topic data type definition

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00007

⌈The data type of a DDS Topic representing a trigger shall be constructed as described in [FO_PRS_DDS_00201].⌋

[SWS_CM_10526] Mapping of Send method

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈When instructed to send a trigger message, the DDS Binding shall construct and send a new sample of the equivalent DDS Topic data type as described in [FO_PRS_DDS_00202].⌋

The DDS serialization rules are defined in section 7.2.3.7.

[SWS_CM_10527] Mapping of Subscribe method

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈When instructed to subscribe to a trigger, the DDS binding shall create a DDS DataReader as described in [FO_PRS_DDS_00203].⌋

[SWS_CM_10528] Creating a DDS DataReader for trigger subscription

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00005, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈The DDS binding shall configure a DDS DataReader for the Topic associated with the `trigger` (see [SWS_CM_10524]), as described in [FO_PRS_DDS_00204].

⌋

[SWS_CM_10529] Defining a DDS DataReaderListener

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈The DDS Binding implementation shall define a `DataReaderListener` object capable of handling notifications when a new sample is received and/or when the matched status of the subscription changes for the Topic data type specified in [SWS_CM_10525], as described in [FO_PRS_DDS_00205].

⌋

[SWS_CM_10530] Mapping of Unsubscribe method

Upstream requirements: RS_CM_00104, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unsubscribe from a service trigger, the DDS binding shall delete the DataReader associated with the `trigger`, as described in [FO_PRS_DDS_00206].⌋

[SWS_CM_10531] Mapping of `GetSubscriptionState()` method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to provide the subscription state, the DDS binding shall query the DataReader associated to the service trigger as described in [FO_PRS_DDS_00207].⌋

[SWS_CM_10532] Mapping of GetNewTriggers method

Upstream requirements: RS_CM_00202, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to get new triggers, the DDS binding shall perform a `take()` on the DataReader without specifying a `max_samples` limit, as described in [FO_PRS_DDS_00208].

After calling `take()`, the binding implementation shall increase the internal trigger count proportionally to the number of samples returned by take().⌋

[SWS_CM_10534] Mapping of SetReceiveHandler method

Upstream requirements: RS_CM_00203, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to register an `TriggerReceiveHandler`, the binding implementation shall link the provided handler to the service trigger's DataReader Listener as described in [FO_PRS_DDS_00209].⌋

[SWS_CM_10535] Mapping of `UnsetReceiveHandler()` method

Upstream requirements: RS_CM_00203, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unregister an `TriggerReceiveHandler`, the binding implementation shall unlink the current handler, if any, as described in [FO_PRS_DDS_00210].⌋

[SWS_CM_10536] Mapping of SetSubscriptionStateHandler method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈When instructed to register a `SubscriptionStateChangeHandler`, the binding implementation shall link the provided handler to the service triggers's DataReader Listener as described in [FO_PRS_DDS_00211].⌋

[SWS_CM_10537] Mapping of UnsetSubscriptionStateHandler method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈When instructed to unregister a `SubscriptionStateChangeHandler`, the binding implementation shall unlink the current handler, if any, as described in [FO_PRS_DDS_00212].⌋

7.2.3.5 Handling Method Calls

The RPC over DDS Specification (DDS-RPC) [23] introduces the concept of DDS Services. These Services provide the mechanisms required to define and implement methods that can be invoked remotely by DDS "client" applications using the building blocks of the DDS data-centric publish-subscribe middleware [20]. In this section, we specify how to handle ara::com method calls over DDS by defining the appropriate mapping between ara::com service methods and DDS service methods.

[SWS_CM_11100] Mapping Methods to DDS Service Methods and Topics

Upstream requirements: RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_-00005, FO_RS_Dds_00008

⌈Every `ServiceInterface` containing one or more `ClientServerOperation`s defined in the role `method` shall have an associated set of DDS Topics as described in [FO_PRS_DDS_00300].⌋

[SWS_CM_11101] DDS Service Request Topic data type definition

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_CM_00200, FO_RS_Dds_00001, FO_RS_Dds_00007

⌈Every `ServiceInterface` containing one or more `ClientServerOperation`s defined in the role `method` shall have an associated DDS Request Topic Type as described in [FO_PRS_DDS_00301].⌋

[SWS_CM_11102] DDS Service Reply Topic data type definition

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_CM_00200, FO_RS_Dds_00001, FO_RS_Dds_00007

⌈Every `ServiceInterface` containing one or more `ClientServerOperation`s defined in the role `method` shall have an associated DDS Reply Topic Type as described in [FO_PRS_DDS_00302].⌋

[SWS_CM_10431] Mapping of ara::core::ErrorCode

Upstream requirements: FO_RS_Dds_00001, FO_RS_Dds_00007

⌈Application-layer errors shall be represented according to the IDL code described in [FO_PRS_DDS_00303]. Since IDL modules are translated to C++ namespaces during IDL to C++ code generation, the additional top-level module `dds` prevents clashing of the generated C++ type with ara::com's own `ara::core::ErrorCode` definition.

⌋

The DDS serialization rules are defined in section 7.2.3.7.

[SWS_CM_11103] Creating a DataWriter to handle method requests on the client side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00015

⌈The DDS binding shall create a DDS DataWriter for the Request Topic associated with all `method`s of the `ServiceInterface` (see [SWS_CM_11101]) upon proxy instantiation, as described in [FO_PRS_DDS_00304].⌋

[SWS_CM_11104] Creating a DataReader to handle method responses on the client side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00016

⌈The DDS binding shall create a DDS DataReader for the Reply Topic associated with all `method`s of the `ServiceInterface` (see [SWS_CM_11102]) upon proxy instantiation, as described in [FO_PRS_DDS_00305].⌋

[SWS_CM_11105] Creating a DataReader to handle method requests on the server side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00016

⌈The DDS binding shall create a DDS DataReader for the Request Topic associated with all `method`s of the `ServiceInterface`) (see [SWS_CM_11101]) upon skeleton instantiation, as described in [FO_PRS_DDS_00306].

The `ServiceSkeleton` constructor `MethodCallProcessingMode` parameter determines the DDS DataReader configuration (for event- or polling-driven operation), as described in [FO_PRS_DDS_00306].

⌋

[SWS_CM_11106] Creating a DataWriter to handle method responses on the server side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00015

⌈The DDS binding shall create a DDS DataWriter for the Reply Topic associated with all `method`s of the `ServiceInterface` (see [SWS_CM_11102]) upon proxy instantiation, as described in [FO_PRS_DDS_00307].⌋

[SWS_CM_11107] Calling a service method from the client side

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈When instructed to call a method from the client side, the DDS binding shall construct a new sample of the Request Topic and send it as described in [FO_PRS_DDS_00308].⌋

The DDS serialization rules are defined in section 7.2.3.7.

[SWS_CM_11108] Notifying the client of a response to a method call

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈To notify the client application of a response as a result of a method call, the DDS binding implementation shall invoke either the `set_value()` operation or the `SetError()` operation of the `ara::core::Promise` corresponding to the `ara::core::Future` that is returned to the caller.

Extraction of result or error values shall be performed as described in [FO_PRS_DDS_00309].

⌋

[SWS_CM_11109] Processing a method call on the server side (event driven)

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈In case a `ara::com::MethodCallProcessingMode==kEvent` or `ara::com::MethodCallProcessingMode==kEventSingleThread` has been passed to the constructor of the `ServiceSkeleton` (see [SWS_CM_00130]), the binding implementation shall create a `DataReaderListener` as described in [FO_PRS_DDS_00310].⌋

[SWS_CM_11110] Creating a DataReaderListener to process asynchronous requests on the server side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈According to [SWS_CM_11105], a `ara::com::MethodCallProcessingMode==kEvent` or `ara::com::MethodCallProcessingMode==kEventSingleThread` of either `kEvent` or `kEventSingleThread` requires the instantiation of a `DataReaderListener` to process asynchronously requests on the server side. This shall be implemented as described in [FO_PRS_DDS_00311].⌋

[SWS_CM_11111] Processing a method call on the server side (polling)

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈In case a `ara::com::MethodCallProcessingMode==kPoll` has been passed to the constructor of the `ServiceSkeleton` (see [SWS_CM_00130]), the `ProcessNextMethodCall` method is to be implemented as described in [FO_PRS_DDS_00312].⌋

[SWS_CM_11112] Sending a method call response from the server side

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈The binding implementation shall send a response upon the return (either as a result of a normal return or through one of the possible `ApApplicationError`s referenced by the `ClientServerOperation` in the role `possibleApError`) of the service method (see [SWS_CM_10306] and [SWS_CM_10307]), as described in [FO_PRS_DDS_00313].⌋

The DDS serialization rules are defined in section 7.2.3.7.

7.2.3.6 Handling Fields

[SWS_CM_11130] Mapping Fields with hasNotifier attribute to DDS Topics

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈The DDS binding shall map every `Field` Notifier defined in the `ServiceInterface` in the role `field` to a DDS Topic, as described in [FO_PRS_DDS_00400].⌋

[SWS_CM_11131] Field Notifier DDS Topic data type definition

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00007

⌈The data type of a DDS Topic representing a `Field` Notifier shall be constructed as described in [FO_PRS_DDS_00401].⌋

The DDS serialization rules are defined in section 7.2.3.7.

[SWS_CM_11132] Mapping of Update method

Upstream requirements: RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈When instructed to send an event message, the DDS Binding shall construct and send a new sample of the equivalent DDS Topic data type as described in [FO_PRS_DDS_00402].⌋

The DDS serialization rules are defined in section 7.2.3.7.

[SWS_CM_11133] Mapping of Subscribe method

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈When instructed to subscribe to a field, the DDS binding shall create a DDS DataReader as described in [FO_PRS_DDS_00403].⌋

[SWS_CM_11134] Creating a DDS DataReader for field subscription

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016, FO_RS_Dds_00005

⌈The DDS binding shall configure a DDS DataReader for the Topic associated with the `field` (see [SWS_CM_11130]), as described in [FO_PRS_DDS_00404].⌋

[SWS_CM_11135] Creating a DDS DataReaderListener for field subscription

Upstream requirements: RS_CM_00103, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005, FO_RS_Dds_00016

⌈The DDS implementation shall define a `DataReaderListener` class to handle field notifications when a new sample is received and/or the matched status of the subscription changes as described in [FO_PRS_DDS_00405].⌋

[SWS_CM_11136] Mapping of Unsubscribe method

Upstream requirements: RS_CM_00104, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈When instructed to unsubscribe from a service field, the DDS binding shall delete the DataReader associated with the `field`, as described in [FO_PRS_DDS_00406].⌋

[SWS_CM_11137] Mapping of GetSubscriptionState method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈The `GetSubscriptionState()` method shall be mapped as specified in [SWS_CM_11022] using the DataReader created in [SWS_CM_11134].⌋

[SWS_CM_11138] Mapping of GetNewSamples method

Upstream requirements: RS_CM_00202, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈The GetNewSamples method shall be mapped as specified in [SWS_CM_11023] using the DataReader created in [SWS_CM_11134].⌋

[SWS_CM_11139] Mapping of GetFreeSampleCount method

Upstream requirements: RS_CM_00202, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈The GetFreeSampleCount method shall be mapped as specified in [SWS_CM_11024] using the DataReader created in [SWS_CM_11134].⌋

[SWS_CM_11140] Mapping of SetReceiveHandler method

Upstream requirements: RS_CM_00203, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈The SetReceiveHandler method shall be mapped as specified in [SWS_CM_11025] using the DataReader created in [SWS_CM_11134].⌋

[SWS_CM_11141] Mapping of `UnsetReceiveHandler()` method

Upstream requirements: RS_CM_00203, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈The `UnsetReceiveHandler()` method shall be mapped as specified in [SWS_CM_11026] using the DataReader created in [SWS_CM_11134].⌋

[SWS_CM_11142] Mapping of SetSubscriptionStateHandler method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈The SetSubscriptionStateHandler method shall be mapped as specified in [SWS_CM_11027] using the DataReader created in [SWS_CM_11134].⌋

[SWS_CM_11143] Mapping of UnsetSubscriptionStateHandler method

Upstream requirements: RS_CM_00106, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈The UnsetSubscriptionStateHandler method shall be mapped as specified in [SWS_CM_11028] using the DataReader created in [SWS_CM_11134].⌋

[SWS_CM_11144] Mapping of Field Get/Set methods to DDS Service Methods and Topics

Upstream requirements: RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈Every `ServiceInterface` containing one or more `Field`s defined in the role `field` with `hasGetter` or `hasSetter` attributes set to `true` shall have an associated set of DDS Topics as described in [FO_PRS_DDS_00414].⌋

[SWS_CM_11145] DDS Service Request Topic data type definition for Field getter and setter operations

Upstream requirements: RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00007

⌈Every `ServiceInterface` containing one or more `Field`s defined in the role `field` with `hasGetter` or `hasSetter` attributes set to `true` shall have an associated DDS Request Topic Type as described in [FO_PRS_DDS_00415].⌋

[SWS_CM_11146] DDS Service Reply Topic data type definition for Field getter and setter operations

Upstream requirements: RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00007

⌈Every `ServiceInterface` containing one or more `Field`s defined in the role `field` with `hasGetter` or `hasSetter` attributes set to `true` shall have an associated DDS Reply Topic Type as described in [FO_PRS_DDS_00416].⌋

[SWS_CM_11147] Creating a DataWriter to handle get/set requests on the client side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈The DDS binding shall create a DDS DataWriter for the Request Topic associated with the getters and setters of the `fields` of the `ServiceInterface` (see [SWS_CM_11145]), as described in [FO_PRS_DDS_00417].⌋

[SWS_CM_11148] Creating a DataReader to handle get/set responses on the client side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈The DDS binding shall create a DDS DataReader for the Reply Topic associated with the getters and setters of the `fields` of the `ServiceInterface` (see [SWS_CM_11145]), as described in [FO_PRS_DDS_00418].⌋

[SWS_CM_11149] Creating a DataReader to handle get/set requests on the server side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈The DDS binding shall create a DDS DataReader for the Request Topic associated with the getters and setters of the `fields` of the `ServiceInterface`) (see [SWS_CM_11146]), as described in [FO_PRS_DDS_00419].

`ServiceSkeleton` constructor `MethodCallProcessingMode` parameter determines the DDS DataReader configuration (for event- or polling-driven operation), as described in [FO_PRS_DDS_00419].⌋

[SWS_CM_11150] Creating a DataWriter to handle get/set responses on the server side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00005

⌈The DDS binding shall create a DDS DataWriter for the Reply Topic associated with the The DDS binding shall create a DDS DataReader for the Reply Topic associated with the getters and setters of the `fields` of the `ServiceInterface`) (see [SWS_CM_11146]), as described in [FO_PRS_DDS_00420].⌋

[SWS_CM_11151] Calling get/set method associated with a field from the client side

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈When instructed to call the `Get()` or `Set()` method associated with a `Field` from the client side, the DDS binding shall construct a new sample of the corresponding Request Topic ans send it as described in [FO_PRS_DDS_00421].⌋

The DDS serialization rules are defined in section 7.2.3.7.

[SWS_CM_11152] Notifying the client of the response to the get/set method call

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈To notify the client application of a response as a result of a method call, the DDS binding implementation shall invoke either the `set_value()` operation or the `SetError()` operation of the `ara::core::Promise` corresponding to the `ara::core::Future` that is returned to the caller.

Extraction of result or error values shall be performed as described in [FO_PRS_DDS_00309].

To notify the client application of a response as a result of call to a `Get()` or `Set()` method associated with a `Field`, the DDS binding implementation shall invoke the `set_value()` operation (see [SWS_CORE_00345] and [SWS_CORE_00346]) with the result data obtained as described in [FO_PRS_DDS_00422].

⌋

[SWS_CM_11153] Processing a get/set method call associated with a field on the server side (event driven)

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00016

⌈In case a `ara::com::MethodCallProcessingMode==kEvent` or `ara::com::MethodCallProcessingMode==kEventSingleThread` has been passed to the constructor of the `ServiceSkeleton` (see [SWS_CM_00130]), the binding implementation shall create a `DataReaderListener` as described in [FO_PRS_DDS_00423].⌋

[SWS_CM_11154] Creating a DataReaderListener to process asynchronous requests for field getters and setters on the server side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈According to [SWS_CM_11149], a `ara::com::MethodCallProcessingMode==kEvent` or `ara::com::MethodCallProcessingMode==kEventSingleThread` `kEvent` or `kEventSingleThread` requires the instantiation of a `DataReaderListener` to process asynchronously requests on the server side. This shall be implemented as described in [FO_PRS_DDS_00424].⌋

[SWS_CM_11155] Processing a get/set method call associated with a field on the server side (polling)

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, FO_RS_Dds_00001, FO_RS_Dds_00008

⌈In case a `ara::com::MethodCallProcessingMode==kPoll` has been passed to the constructor of the `ServiceSkeleton` (see [SWS_CM_00130]), the `ProcessNextMethodCall` method shall be implemented as described in [FO_PRS_DDS_00425].⌋

[SWS_CM_11156] Sending a response for a get/set method call associated with a field from the server side

Upstream requirements: RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, FO_RS_Dds_00001, FO_RS_Dds_00008, FO_RS_Dds_00015

⌈The binding implementation shall send a response upon the return of (1) a `SetHandler` in the case of a `Set()` operation; (2) a `GetHandler` in the case of a `Get()` operation where a `GetHandler` has previously been registered; or (3) a lookup operation[5] as a result of a `Get()` operation where no `GetHandler` was previously registered.

This shall be performed as decribed in [FO_PRS_DDS_00426].⌋

The DDS serialization rules are defined in section 7.2.3.7.

7.2.3.7 Serialization of Payload

[SWS_CM_11040] DDS standard serialization rules

Upstream requirements: RS_CM_00204, RS_CM_00201, FO_RS_Dds_00001, FO_RS_Dds_00002

⌈The serialization of the payload shall be done according to the DDS standard serialization rules defined in section 7.4.3.5 of [22].⌋

[5] An internal lookup operation to retrieve the current value of a field.

7.2.3.7.1 Basic Data Types

[SWS_CM_11041] DDS serialization of `StdCppImplementationDataType` of `category` `VALUE`

Upstream requirements: RS_CM_00204, RS_CM_00200, RS_CM_00102, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈`StdCppImplementationDataType` of `category` `VALUE` shall be serialized, as described in [FO_PRS_DDS_00501], according to the standard serialization rules for the equivalent DDS `PRIMITIVE_TYPE` defined in section 7.4.3.5 of [22].⌋

7.2.3.7.2 Enumeration Data Types

[SWS_CM_11042] DDS serialization of enumeration data types

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Enumeration data types shall be serialized, as described in [FO_PRS_DDS_00502].⌋

7.2.3.7.3 Structured Data Types (structs)

[SWS_CM_11043] DDS serialization of `StdCppImplementationDataType` of `category` `STRUCTURE`

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈`StdCppImplementationDataType` of `category` `STRUCTURE` shall be serialized as described in [FO_PRS_DDS_00503].⌋

7.2.3.7.4 Strings

[SWS_CM_11044] DDS serialization of `StdCppImplementationDataType` of `category` `STRING` with string `shortName`

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈An `StdCppImplementationDataType` of `category` `STRING` shall be serialized, as described in [FO_PRS_DDS_00504].⌋

[SWS_CM_11046] Encoding Format and Endianness of Strings in DDS

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈Section 7.4.1.1.2 of [22] specifies the standard character encoding format for `STRING_TYPE`: UTF-8. The serialized version shall be as described in [FO_PRS_DDS_00505].⌋

7.2.3.7.5 Vectors and Arrays

[SWS_CM_11047] DDS serialization of `CppImplementationDataType` of `category` `VECTOR`

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈A `CppImplementationDataType` of `category` `VECTOR` shall be serialized, as described in [FO_PRS_DDS_00506].⌋

[SWS_CM_11048] DDS serialization of `CppImplementationDataType` of `category` `ARRAY`

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈A `CppImplementationDataType` of `category` `ARRAY` shall be serialized, as described in [FO_PRS_DDS_00507].⌋

7.2.3.7.6 Associative Maps

[SWS_CM_11049] DDS serialization of `CppImplementationDataType` of `category` `ASSOCIATIVE_MAP`

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈`CppImplementationDataType` of `category` `ASSOCIATIVE_MAP` shall be serialized, as described in [FO_PRS_DDS_00508].⌋

7.2.3.7.7 Variant

[SWS_CM_11050] DDS serialization of `CppImplementationDataType` of `category` `VARIANT`

Upstream requirements: RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, FO_RS_Dds_00001, FO_RS_Dds_00002, FO_RS_Dds_00007

⌈`CppImplementationDataType` of `category` `VARIANT` shall be serialized, bas described in [FO_PRS_DDS_00509].⌋

7.3 Security

In the following chapter the behavior according to the meta-model of access control and secure communication shall be described.

7.3.1 IAM

Access control for Communication Management allows to restrict the instances and elements of services that a local application or a *remote subject* (e.g., a remote ECU) may request to access. Having access control in place reduces the potential damage that a compromised application (in case of local IAM) or a compromised ECU (in case of remote IAM) can cause.

Figure 7.10 demonstrates an example scenario where local IAM and remote IAM can take place. Upon a method call from a service, the client's request will be checked by the local IAM to ensure that the application is issuing a legitimate request based on its configured access rights. After successful authorization, the request will be forwarded to the machine where the service is running. When the request arrives at the recipient machine, the remote IAM takes place and a check will be performed to verify if such a request coming from the given sender ECU was envisioned.

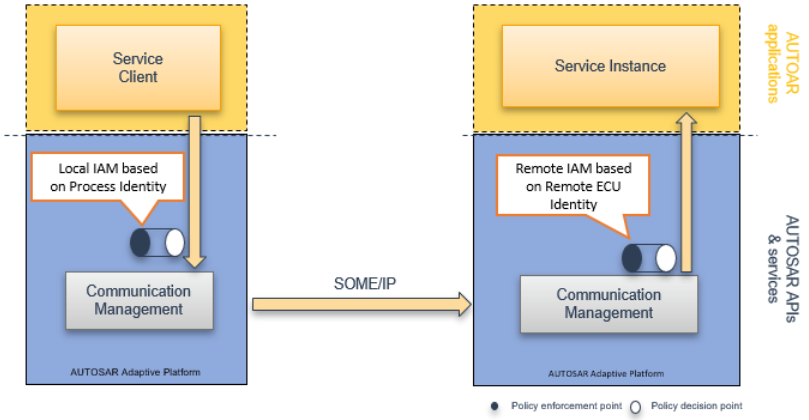

Figure 7.10: Local and Remote Identity and Access Management

The following assumption has to be held true to realize access control:

1. Communication between two applications has to be realized by using ara::com interfaces Communication Management to enable access control.

All access permissions for Communication Management are modeled using `ComGrant` model elements. A `ComGrant` can be used to model access permissions that either apply to a `Machine`-local `Process` or to a remote subject, i.e., either a local Process or a remote entity can be the *subject* of the access control policy: If a `ComGrant` references an `AbstractIamRemoteSubject` in the role `remoteSubject`, then the subjects of the `ComGrant` are all remote entities that can be identified using the information specified in the referenced `AbstractIamRemoteSubject`. If a `ComGrant` does not reference any `remoteSubject`, then the subjects of the `ComGrant` are all `Process`es referenced in the role `process` by `ServiceInstanceToPortPrototypeMapping`s which reference an `AdaptivePlatformServiceInstance` in the role `serviceInstance` that is referenced by the `ComGrant` in the role `serviceInstance`.

Local access control and remote access control may be enforced independently from each other.

7.3.1.1 Configuration of Access Control

Depending on the architecture and the security model, all local `Process`es might be trusted, thus not requiring local access control. Furthermore, it is possible that all remote ECUs are trusted, e.g., because access control is already performed locally. For these cases, there are two configuration options to enable remote access control and local access control independently.

[SWS_CM_10493] Local Access Control Activation

Upstream requirements: RS_IAM_00002

⌈If `CmModuleInstantiation.localComAccessControlEnabled` is defined and is set to false, CM shall perform no local access control, i.e., no access to any service from a local `Process` shall be restricted because of missing `ComGrant`s. If `CmModuleInstantiation.localComAccessControlEnabled` is not defined or is set to true, CM shall perform local access control.⌋

[SWS_CM_10494] Remote Access Control Activation

Upstream requirements: RS_IAM_00002

⌈If `CmModuleInstantiation.remoteAccessControlEnabled` is defined and is set to false, CM shall perform no remote access control, i.e., no access to any service from a remote subject shall be restricted because of missing `ComGrant`s. If `CmModuleInstantiation.remoteAccessControlEnabled` is not defined or is set to true, CM shall perform remote access control.⌋

[SWS_CM_10542] Local access control on providing service instances

Upstream requirements: RS_IAM_00001, RS_IAM_00002

⌈If a `Process` requests to provide a service instance or any element thereof, but there exists no `ComOfferServiceGrant` that

- does not reference any remote subject in the role `remoteSubject` and
- references the requested `ProvidedApServiceInstance` in the role `serviceInstance` and the `ProvidedApServiceInstance` is referenced by a `ServiceInstanceToPortPrototypeMapping` in the role `serviceInstance` and the `ServiceInstanceToPortPrototypeMapping` references the requesting `Process` in the role `process`,

then Communication Management shall drop the request.⌋

[SWS_CM_90006] Local access control on service discovery

Upstream requirements: RS_IAM_00006, RS_IAM_00007, RS_IAM_00010

⌈If a `Process` requests to find a service, but there exists no `ComGrant` that

- does not reference any remote subject in the role `remoteSubject` and
- references the requested `RequiredApServiceInstance` in the role `serviceInstance` and the `RequiredApServiceInstance` is referenced by a `ServiceInstanceToPortPrototypeMapping` in the role `serviceInstance` and the `ServiceInstanceToPortPrototypeMapping` references the requesting `Process` in the role `process`,

then Communication Management shall drop the request and the constructor of the `ServiceProxy` class (see [SWS_CM_00131], [SWS_CM_10438]) shall handle the error as a `InsufficientPermissionsViolation`.⌋

[SWS_CM_90001] Local access control on executing methods

Upstream requirements: RS_IAM_00006, RS_IAM_00007, RS_IAM_00010

⌈If a Process executes a method of a service interface, but there exists no ComMethodGrant that

- does not reference any remote subject in the role remoteSubject and
- references the requested RequiredApServiceInstance in the role serviceInstance and the RequiredApServiceInstance is referenced by a ServiceInstanceToPortPrototypeMapping in the role serviceInstance and the ServiceInstanceToPortPrototypeMapping references the requesting Process in the role process,
- references the requested method in the role serviceDeployment,

then Communication Management shall drop the request and the method operator() shall handle the error as a InsufficientPermissionsViolation.⌋

Note:
In [SWS_CM_90001], field getters and setters are also methods.

[SWS_CM_90003] Local access control on receiving events

Upstream requirements: RS_IAM_00006, RS_IAM_00007, RS_IAM_00010

⌈If a Process subscribes to an event of a service interface, but there exists no ComEventGrant that

- does not reference any remote subject in the role remoteSubject and
- references the requested RequiredApServiceInstance in the role serviceInstance and the RequiredApServiceInstance is referenced by a ServiceInstanceToPortPrototypeMapping in the role serviceInstance and the ServiceInstanceToPortPrototypeMapping references the requesting Process in the role process,
- references the subscribed event in the role serviceDeployment,

then Communication Management shall drop the request and the Subscribe() method of the respective Event class shall handle the error as a InsufficientPermissionsViolation.⌋

Note:
In [SWS_CM_90003], field notifiers are also considered as events.

[SWS_CM_10539] Local access control on receiving triggers

Upstream requirements: RS_IAM_00006, RS_IAM_00007, RS_IAM_00010

⌈If a Process subscribes to a trigger of a service interface, but there exists no ComTriggerGrant that

- does not reference any remote subject in the role `remoteSubject` and
- references the requested `RequiredApServiceInstance` in the role `serviceInstance` and the `RequiredApServiceInstance` is referenced by a `ServiceInstanceToPortPrototypeMapping` in the role `serviceInstance` and the `ServiceInstanceToPortPrototypeMapping` references the requesting `Process` in the role `process`,
- references the subscribed trigger in the role `serviceDeployment`,

then Communication Management shall drop the request and the `Subscribe()` method of the respective `Trigger` class shall handle the error as a `InsufficientPermissionsViolation`.⌋

Note:
In case of [SWS_CM_90003] dropping data, the application will not be notified.

A logging facility for security events is currently not defined in the AUTOSAR Adaptive Platform. Logging violations of access restrictions according to [SWS_CM_90001], [SWS_CM_90003], [SWS_CM_10542], [SWS_CM_10543] and [SWS_CM_90006] is up to the implementation or specific ECU projects.

7.3.1.2 Remote Access Control

In order to enforce access control on remote entities, the requesting entity first has to be authenticated, i.e., the identity of the *remote subject* has to be established. Then, it has to be decided whether the access is allowed according to the modeled grants.

There are currently three ways to authenticate a remote subject:

- **TLS:** If the remote subject is connected via (D)TLS secure communication, properties of this TLS connection and the used certificates can be used for authenticating the remote subject.
- **IPsec:** If IPsec is used to establish secure communication, IP related information specified for IPsec configuration can be used for authenticating the remote subject.
- **IP:** If IP based communication is used and the authenticity of communication partners can be guaranteed by, e.g., the operational environment, IP related information can be used for authenticating the remote subject.

Please note that while `SecOC` can also provide authenticity of a communication partner, it is not used in this section, because the existing association between `SecOC` keys and DataIDs already provides a fine grained access control mechanism directly on the level of secure communication and thus additionally applying IAM would not yield any benefit.

[SWS_CM_10495] TLS-based Authentication

Upstream requirements: RS_CM_00803

⌈Communication Management shall associate remote subjects communicating via an established (D)TLS connection to a `TlsIamRemoteSubject` according to [TPS_MANI_03240].⌋

[SWS_CM_10496] IP and IPsec-based Authentication

Upstream requirements: RS_CM_00803

⌈Communication Management shall associate remote subjects communicating via IP to an `IPSecIamRemoteSubject` or an `IpIamRemoteSubject` according to [TPS_MANI_03242] and [TPS_MANI_03244].⌋

Please note that IPsec is usually handled by the OS and may therefore be transparent to Communication Management. Therefore, authentication of IPsec secured connections relies on tuples of IP addresses, protocols, and ports only.

[SWS_CM_10497] Authentication Failure

Upstream requirements: RS_CM_00803

⌈If `CmModuleInstantiation.remoteAccessControlEnabled` is set to true and a remote subject cannot be authenticated, Communication Management shall silently drop all messages from this remote subject.⌋

[SWS_CM_10543] Remote access control on providing service instances

Upstream requirements: RS_IAM_00001, RS_IAM_00002

⌈If a remote subject provides a service instance or any element thereof, but there exists no `ComOfferServiceGrant` that

- references the providing remote subject in the role `remoteSubject` and
- references the provided `RequiredApServiceInstance` in the role `serviceInstance`,

then Communication Management shall drop all requests to and from this service instance.⌋

Note:
The remote subject can be identified through the unicast endpoint of the service offer message.

[SWS_CM_10498] Remote access control on executing methods

Upstream requirements: RS_IAM_00001, RS_IAM_00002

⌈If a remote subject requests the execution of a method of a service interface, but there exists no `ComMethodGrant` that

- references the requesting remote subject in the role `remoteSubject` and
- references a `ProvidedApServiceInstance` in the role `serviceInstance` and
- references the requested method in the role `serviceDeployment`,

then Communication Management shall drop the request.⌋

Note:
In [SWS_CM_10498], field getters and setters are also methods.

[SWS_CM_10501] Remote access control on consuming events

Upstream requirements: RS_IAM_00001, RS_IAM_00002

⌈If a remote subject subscribes to an event of a service interface, but there exists no `ComEventGrant` that

- references the subscribing remote subject in the role `remoteSubject` and
- references a `ProvidedApServiceInstance` in the role `serviceInstance` and
- references the subscribed event in the role `serviceDeployment`,

then Communication Management shall drop the subscription request.⌋

Note:
In [SWS_CM_10501], field notifiers are also considered as events.

[SWS_CM_10541] Remote access control on consuming triggers

Upstream requirements: RS_IAM_00001, RS_IAM_00002

⌈If a remote subject subscribes to an `trigger` of a service interface, but there exists no `ComTriggerGrant` that

- references the subscribing remote subject in the role `remoteSubject` and
- references a `ProvidedApServiceInstance` in the role `serviceInstance` and
- references the `ServiceEventDeployment` in the role `serviceDeployment` that in turn references the subscribed `trigger`.

then Communication Management shall drop the subscription request.⌋

[SWS_CM_10505] Remote access control on consuming field notifiers

Upstream requirements: RS_IAM_00001, RS_IAM_00002

⌈If a remote subject subscribes to a field notifier , but there exists no `ComFieldGrant` that

- references the subscribing remote subject in the role `remoteSubject` and
- references a `ProvidedApServiceInstance` in the role `serviceInstance` and
- references the event in the role `serviceDeployment`,

then Communication Management shall drop the the subscription request.⌋

[SWS_CM_10506] Remote access control on calling field setters

Upstream requirements: RS_IAM_00001, RS_IAM_00002

⌈If a remote subject requests the execution of a set method of a field, but there exists no `ComFieldGrant` that

- has the parameter `ComFieldGrant.role` set to `setter` or `getterSetter` and
- references the requesting remote subject in the role `remoteSubject` and
- references a `ProvidedApServiceInstance` in the role `serviceInstance` and
- references the event in the role `serviceDeployment`,

then Communication Management shall drop the request.⌋

[SWS_CM_10507] Remote access control on calling field getters

Upstream requirements: RS_IAM_00001, RS_IAM_00002

⌈If a remote subject requests the execution of a get method of a field, but there exists no `ComFieldGrant` that

- has the parameter `ComFieldGrant.role` set to `getter` or `getterSetter` and
- references the requesting remote subject in the role `remoteSubject` and
- references a `ProvidedApServiceInstance` in the role `serviceInstance` and
- references the event in the role `serviceDeployment`,

then Communication Management shall drop the request.⌋

7.3.2 Secure Communication

Communication in Adaptive Platform can be transported via TCP and UDP. Therefore different security mechanisms have to be available to secure the communication. The following security protocols are currently supported:

- DDS Security
- SecOC
- TLS 1.2 (see [24])
- DTLS 1.2 (see [25])
- IPSec
- MACsec

The configuration of SecOc and TLS security protocols has a dependency on the network binding:

- For SOME/IP network binding AUTOSAR allows the configuration of secure communication for a ServiceInterface by configuring either `TlsSecureComProps` meta-class or `SecOcSecureComProps` meta-class . Both are specialization of `SecureComProps` class that is referenced by `ServiceInstanceToMachineMapping`. In the case of SecOc additionally `ServiceInterfaceElementSecureComConfig` needs to be defined and it determines the configuration settings for the individual ServiceInterface elements. When `TlsSecureComProps` is configured, all the service interface elements are secured and `ServiceInterfaceElementSecureComConfig` is not used.
- For Signal based network binding, only SecOc configuration is possible, and the configuration is determined by `SecureCommunicationAuthenticationProps` of a SecuredIPdu referenced by the PduTriggering. `SecureComProps` is not used in the context of signal-based network binding.
- For DDS Network binding, DDS Transport Security over TCP (TLS), DDS Transport Security over UDP (DTLS) and DDS Security [26] (as transport-independent security) are valid, independent and mutually exclusive choices for securing underlying DDS communications.

The configuration of Ipsec (IPSecConfig) in aggregated by a NetworkEndpoint therefore it is independent of the network binding.

SOME/IP supports one-to-many (unicast) and many-to-many (multicast) communication paradigms. These paradigms may switch at runtime for events (see `multicastThreshold`).

It is therefore important to be aware of the limitations of the secure channel approach:

- **Confidentiality of events**
  If events are transported using UDP and may be sent using multicast, they cannot be guaranteed confidential due to the fact that only SecOC can be used to secure multicast communication and SecOC does not offer confidentiality. This restriction does not apply to DDS Security.

7.3.2.1 Creation and use of secure channels

7.3.2.1.1 SOME/IP and DDS network binding

[SWS_CM_90101] Secure UDP and TCP channel creation for TLS, DTLS and SecOC

Upstream requirements: RS_CM_00801

⌈The Communication Management software shall create secure UDP channels according to the input for all `SecureComProps` referenced by `ServiceInstanceToMachineMapping` in the role `secureComPropsForUdp`. The Communication Management software shall create secure TCP channels according to the input for all `SecureComProps` referenced by `ServiceInstanceToMachineMapping` in the role `secureComPropsForTcp`. Secure channels may be shared by multiple `AdaptivePlatformServiceInstance`s by multiplexing the communication, i.e. by referencing the same `SecureComProps` in the same role.⌋

[SWS_CM_90102] Using secure TLS, DTLS and SecOC channels

Upstream requirements: RS_CM_00801, RS_CM_00803

⌈All communication triggered by a `Skeleton` or `Proxy` shall be sent via the respective secure channel according to the configuration input. In the configuration the appropriate secure channel is identified by examining the references to `SecureComProps` of `ServiceInstanceToMachineMapping` for the `AdaptivePlatformServiceInstance` that is mapped to an `EthernetCommunicationConnector` of a `Machine` by this `ServiceInstanceToMachineMapping`.⌋

The actual secure channel to be created is determined by the concrete sub-class of the `SecureComProps` base-class.

[SWS_CM_90201] Secure TLS and DTLS channel creation in the DDS Network Binding

Upstream requirements: RS_CM_00801

⌈Secure channels shall be created as specified in [SWS_CM_90101].⌋

[SWS_CM_90202] Using TLS and DTLS secure channels in the DDS Network Binding

Upstream requirements: RS_CM_00801, RS_CM_00803

⌈Secure channels shall be used as specified in [SWS_CM_90102].⌋

7.3.2.2 DDS Security

DDS Security, as defined in [26], is a complementary standard to DDS, providing transport-independent security measures (authentication, secrecy, non-repudiation, integrity, access control and logging) without requiring changes to application logic.

Mapping DDS Service Interface and Instance Deployment models, as well as IAM Communications Grant models, to DDS QoS policies, and DDS Security certificate, governance and permission files is defined by [27].

[SWS_CM_90218] Enforcement of IAM grants through DDS Security

Upstream requirements: RS_IAM_00001, RS_IAM_00002, FO_RS_Dds_00001, FO_RS_Dds_00009

⌈Adaptive Applications providing or requiring Service Interface Instances through the DDS Network Binding shall enforce, when provided, deployed DDS Security policies.⌋

7.3.2.3 SecOC

The Secure Onboard Communication (SecOC) feature is embedded into the Adaptive Communication Management. The behavioral aspects of the SecOC protocol are specified in the *PRS_SecOcProtocolSpecification*.

One major goal is to achieve interoperability with the AUTOSAR Classic Platform *SecOC* functionality. This is especially applicable to the usage of *UDP multicast* messages (where SecOC is currently the only protocol supported) and secured signal-based communication with AUTOSAR Classic Platform through the signal-based network binding.

The SecOC secure channel may provide authenticity and integrity.

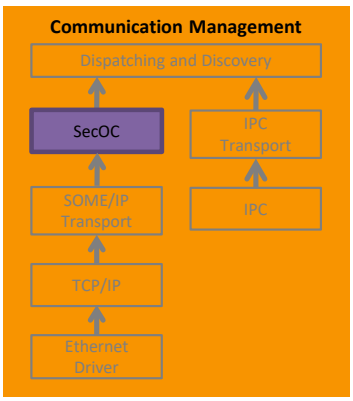

Figure 7.11: SecOC embedded in the Adaptive Communication Management

In order to achieve interoperability with the AUTOSAR Classic Platform the SecOC should be applied identically also in Adaptive Communication Management. The authentication information comprises of an Authenticator (e.g. Message Authentication Code) and optionally a Freshness Value.

The SOME/IP Message Header as shown in figure 7.12 divided into two parts: Part I containing the Message ID and the Length and Part II containing Request ID, Protocol Version, Interface Version, Message Type and Return Code(SOME/IP Protocol Specification [4]).

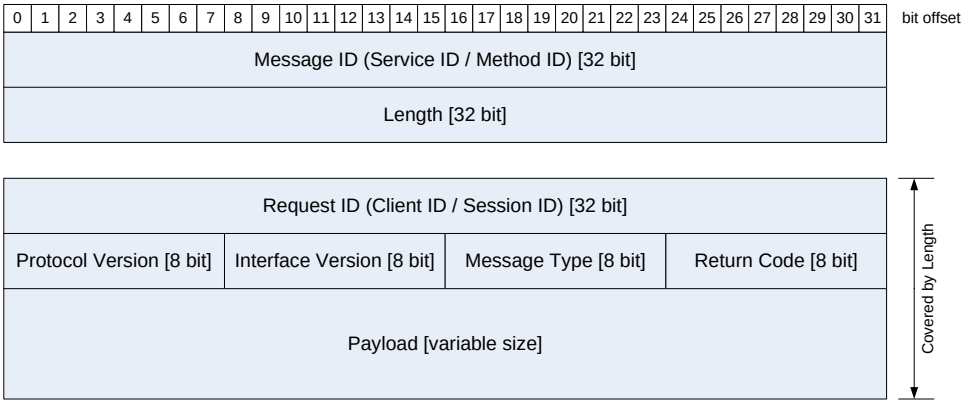

Figure 7.12: SOME/IP header structure

In figure 7.14 the handling of the SOME/IP payload, the SecOC part, and the SOME/IP Message Header are illustrated. This setup is defined by the AUTOSAR Classic platform. In order to achieve interoperability the Communication Management shall implement an identical behavior. It is essential that the Part I of the SOME/IP Message header is NOT covered by the SecOC calculation.

To keep the interoperability with the AUTOSAR Classic Platform and provide the optional Freshness Value Management functionality the Adaptive Communication Management will rely on a pluggable Freshness Value Management Library.

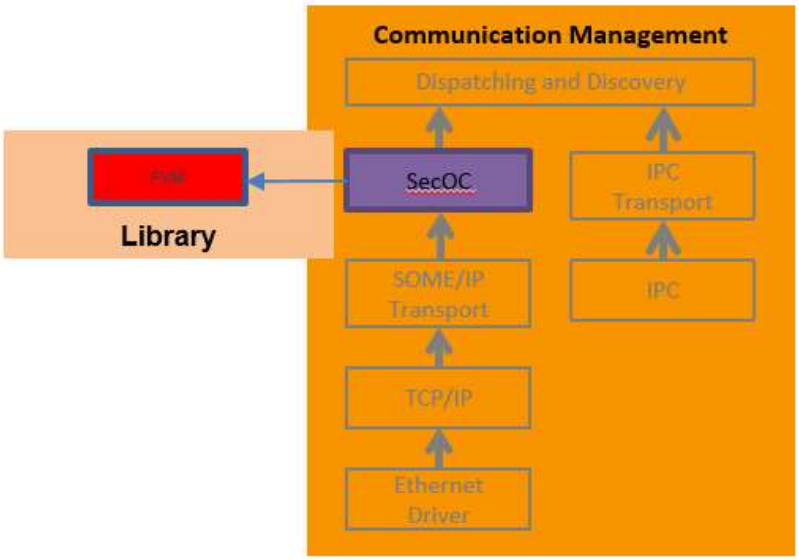

Figure 7.13: Freshness Value Management Pluggable Library

This library will provide the Freshness Value Management API comprising the replica of the AUTOSAR Classic Platform *FreshnessManagement* Client Server Interface and corresponding functions of the *Callout Definitions*.

7.3.2.3.1 SOME/IP network binding

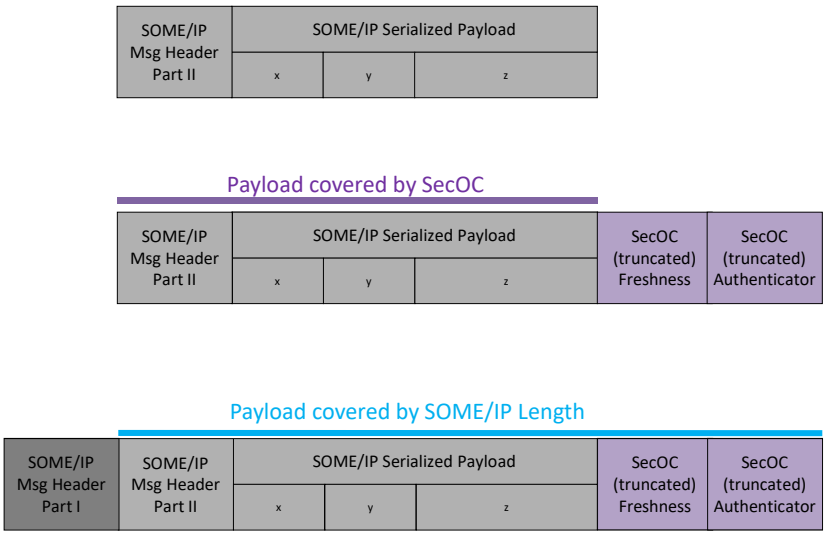

Figure 7.14: Payload covered by SecOC and SOME/IP transport

[SWS_CM_90108] SecOC secure channel for methods using reliable transport

Upstream requirements: RS_CM_00801

⌈A SecOC secure channel shall be created and used if:

- A SecOcSecureComProps instance is referenced in the role secureComPropsForTcp by a ServiceInstanceToMachineMapping and a Method of the AdaptivePlatformServiceInstance is selected for transmission over the secured channel by the ServiceInterfaceElementSecureComConfig and this Method of the AdaptivePlatformServiceInstance is configured for transmission over “tcp” by transportProtocol in the associated SomeipMethodDeployment.

⌋

[SWS_CM_90115] SecOC secure channel for methods using unreliable transport

Upstream requirements: RS_CM_00801

⌈A SecOC secure channel shall be created and used if:

- A SecOcSecureComProps instance is referenced in the role secureComPropsForUdp by a ServiceInstanceToMachineMapping and a Method of the AdaptivePlatformServiceInstance is selected for transmission over the secured channel by the ServiceInterfaceElementSecureComConfig and this Method of the AdaptivePlatformServiceInstance is configured for transmission over “udp” by transportProtocol in the associated SomeipMethodDeployment.

⌋

[SWS_CM_90109] SecOC secure channel for events and triggers using reliable transport

Upstream requirements: RS_CM_00801

⌈A SecOC secure channel shall be created and used if:

- A SecOcSecureComProps instance is referenced in the role secureComPropsForTcp by a ServiceInstanceToMachineMapping and an event or trigger of the AdaptivePlatformServiceInstance is selected for transmission over the secured channel by the ServiceInterfaceElementSecureComConfig and this event or trigger of the AdaptivePlatformServiceInstance is configured for transmission over “tcp” by transportProtocol in the associated SomeipEventDeployment.

⌋

[SWS_CM_90116] SecOC secure channel for events and triggers using unreliable transport

Upstream requirements: RS_CM_00801

⌈A SecOC secure channel shall be created and used if:

- A `SecOcSecureComProps` instance is referenced in the role `secureComPropsForUdp` by a `ServiceInstanceToMachineMapping` and an `event` or `trigger` of the `AdaptivePlatformServiceInstance` is selected for transmission over the secured channel by the `ServiceInterfaceElementSecureComConfig` and this `event` or `trigger` of the `AdaptivePlatformServiceInstance` is configured for transmission over “udp” by `transportProtocol` in the associated `SomeipEventDeployment`.

⌋

[SWS_CM_90110] SecOC secure channel for fields

Upstream requirements: RS_CM_00801

⌈The requirements [SWS_CM_90108], [SWS_CM_90109], [SWS_CM_90115], [SWS_CM_90116] apply to fields in the same manner, since fields are a composition of `methods` and `events`.⌋

[SWS_CM_11271] SecOC secure channel behavior

Upstream requirements: RS_CM_00801

⌈Whenever a `SecOC` secure channel interaction is detected (based on the configuration options of [SWS_CM_90108], [SWS_CM_90115], [SWS_CM_90109], [SWS_CM_90116], and [SWS_CM_90110]) the `SecOC` functionality shall be applied according to:

- sending according to [SWS_CM_11274], [SWS_CM_11275]
- reception according to [SWS_CM_11276], [SWS_CM_11277]

⌋

[SWS_CM_11272] Lifecycle management of FVM

Upstream requirements: RS_CM_00801

⌈The lifecycle of an `SecOC FreshnessValueManager` implementation shall be managed by ara::com.⌋

[SWS_CM_11273] Initialization of the FVM

Upstream requirements: RS_CM_00801

⌈

- The SecOC FreshnessValueManager implementation shall be initialized by calling Freshness Value Management Library API apext::com::secoc::FVM::Initialize.

⌋

[SWS_CM_11274] SecOC secure channel sending

Upstream requirements: RS_CM_00801, RS_CM_00802

⌈If a message is configured to be SecOC sent, the message shall be secured according to [28] and following steps shall be performed:

- the message shall be handled as Authentic message by the Communication Management
- the message Authentication shall be performed in the order of operations after the E2E protection calculations
- if the ServiceInterfaceElementSecureComConfig has an attribute freshnessValueId defined, the Communication Management shall call the Freshness Value Mananement Library API apext::com::secoc::FVM::GetTxFreshness with the freshnessValueId
- calculate the MAC using the Authentic message ([PRS_SecOc_00200] see [28]), (optionally the Freshness Value), and the dataId
- if the attribute authInfoTxLength is defined, the Authenticator ([PRS_SecOc_00210] see [28]) shall be truncated
- if the attribute freshnessValueTxLength is defined, the Freshness Value shall be truncated ([PRS_SecOc_00103] see [28])
- combine the Authentic message, (truncated) Freshness Value, and (truncated) Authenticator ([PRS_SecOc_00211] see [28])
- continue in the Communication Management with the send processing

The details for the construction of secure message are described in: [PRS_SecOc_00103], [PRS_SecOc_00105], [PRS_SecOc_00200], [PRS_SecOc_00208], [PRS_SecOc_00210], [PRS_SecOc_00211] (see [28])⌋

[SWS_CM_11275] SecOC secure message build attempts

Upstream requirements: RS_CM_00801

⌈For every message to be sent and secured with SecOC [28] an Authentication Build Counter shall be maintained:

- the Authentication Build Counter shall be set to 0 if the operation was successful.

- if the query of the freshness value `apext::com::secoc::FVM::GetTxFreshness` return a recoverable error `kFVNotAvailable`, or an error occurs during calculation of the `Authenticator`, the `Authentication Build Counter` is incremented and the process of securing the message will be retried in an implementation specific manner.
- if the `Authentication Build Counter` has reached the `SecOC` implementation specific threshold `SecOCAuthenticationBuildAttempts`, the message shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation).

⌋

[SWS_CM_11276] SecOC secure channel reception

Upstream requirements: RS_CM_00801, RS_CM_00802

⌈If a message is configured to be `SecOC` received and the attribute `securedRxVerification` is set to true or is not defined, then the message shall be verified according to [28] and following steps shall be performed:

- the message shall be handled as Secured message by the Communication Management
- if the attribute `freshnessValueTxLength` is defined, the `Freshness Value` will be calculated by calling the `Freshness Value Mananement Library API` `apext::com::secoc::FVM::GetRxFreshness` with `SecOCFreshnessValueID` equals to defined `freshnessValueId` and with the `SecOCTruncatedFreshnessValue` equals to the extracted `Truncated Freshness Value`([PRS_SecOc_00317] see [28]) from the Secured message, otherwise the `Freshness Value`([PRS_SecOc_00316] see [28]) shall be extracted from the Secured message itself
- if the attribute `authInfoTxLength` is defined, the `Truncated Authenticator` shall be extracted from the Secured message, otherwise the `Authenticator`([PRS_SecOc_00317] see [28]) shall be extracted from the Secured message
- verify the message by calculating the MAC using the Secured message, optionally the `Freshness Value`([PRS_SecOc_00300], and comparing the result with received `Truncated Authenticator` and continue in the Communication Management with the receive processing
- the message authentication procedure is done before E2E checks

The details for the verification of secure message are described in: [PRS_SecOc_00103], [PRS_SecOc_00300], [PRS_SecOc_00316], [PRS_SecOc_00317], [PRS_SecOc_00330] (see [28])⌋

[SWS_CM_11372] SecOC secure channel reception bypass

Upstream requirements: RS_CM_00801

⌈If a message is configured to be SecOC received and the attribute securedRxVerification is set to false, then

- the message shall be handled as Secured message without verification by the Communication Management
- the Authentic message part shall be extracted and processed
- the VerificationStatus shall be set to VerificationStatusResult.kSecOcNoVerification

⌋

[SWS_CM_11277] SecOC secure message verification attempts

Upstream requirements: RS_CM_00801

⌈For every message received and secured with SecOc, an Authentication Build Counter shall be maintained:

- the Authentication Build Counter shall be set to 0 if the operation was successful.
- if the query of the freshness value Freshness Value Mananement Library API apext::com::secoc::FVM::GetRxFreshness returns a recoverable error kFVNotAvailable, or an error occurs during calculation of the Authenticator, the Authentication Build Counter shall be incremented and the process of message verification will be retried in an implementation specific manner.
- if the counter has reached the parameter authenticationRetries, the message shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation).
- if the calculation of the Authenticator was successful but the verification failed for the parameter authenticationRetries([PRS_SecOc_00306] see [28]), the message shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation).

The process is described in: [PRS_SecOc_00306], [PRS_SecOc_00309] (see [28])⌋

The SecOC VerificationStatus service is used to propagate the status of each verification attempt from the SecOC to an application. It can be used to continuously monitor the number of failed verification attempts and would allow setting up a security management system/intrusion detection system that is able to detect an attack flood and react with adequate dynamic countermeasures.

[SWS_CM_11278] SecOC verification results

Upstream requirements: RS_CM_00801

⌈Communication Management shall make each verification result (`VerificationStatusResult`) accessible via the `VerificationStatus` service.⌋

[SWS_CM_11279] SecOc override the verification result

Upstream requirements: RS_CM_00801

⌈Communication Management shall allow the configuration of `SecOC` behavior via the `VerifyStatusOverride` or `VerifyStatusOverride` methods. The overwrite options are defined by `OverrideStatus`. The configuration is available per `dataID` in the case of `VerificationStatusConfigurationByDataId` service or per `freshnessID` in the case of `VerificationStatusConfigurationByFreshnessId` service.⌋

7.3.2.3.2 Signal based network binding

The SOME/IP Message Header as shown in figure 7.12 is divided into two parts: Part I containing the Message ID and the Length and Part II containing Request ID, Protocol Version, Interface Version, Message Type and Return Code (SOME/IP Protocol Specification [4]).

In case of signal-service-translation only a partial header is used, namely the Part I. In figure 7.15 the handling of the Header Part I, the signal based payload, and the SecOC part is illustrated.

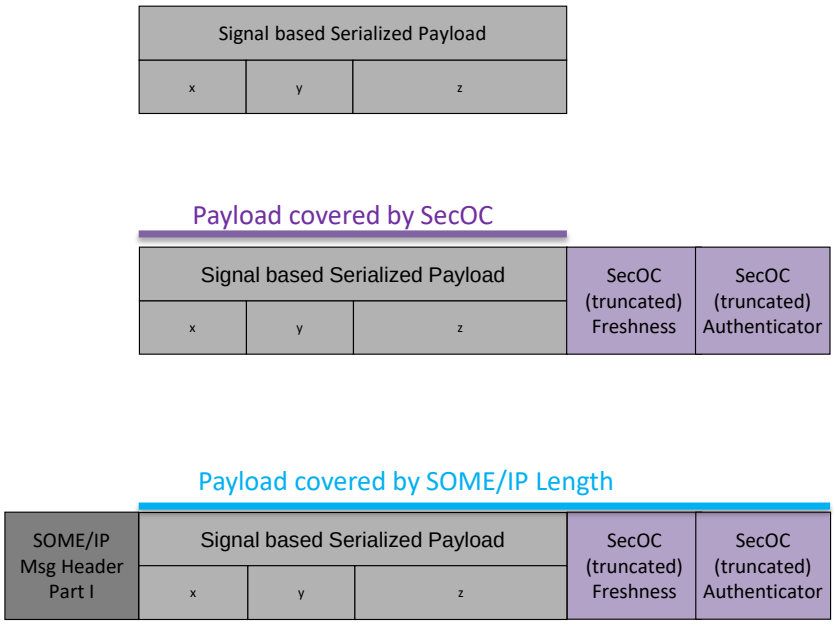

Figure 7.15: Payload covered by SecOC and Signal2Service transport

[SWS_CM_11346] Usage of SecOC configuration with Signal Based Network Binding

Upstream requirements: RS_CM_00801

⌈If the ISignalTriggering is used in a signal-service-translation (the attribute SomeipEventDeployment.serializer equals signalBased), CM shall check if the PduTriggering of this ISignalIPdu is referenced by a SecuredIPdu and use the SecureCommunicationAuthenticationProps, SecureCommunication-FreshnessProps and SecureCommunicationProps of the SecuredIPdu as configuration of SecOc.⌋

As described in Security chapter of [5], in the context of signal-based communication, SecOC is highly embedded into the Classic platform architecture the signal-service translation approach on security is to use the same architecture for its specification.

The input for signal based SecOC configuration is shown in figure 7.16:

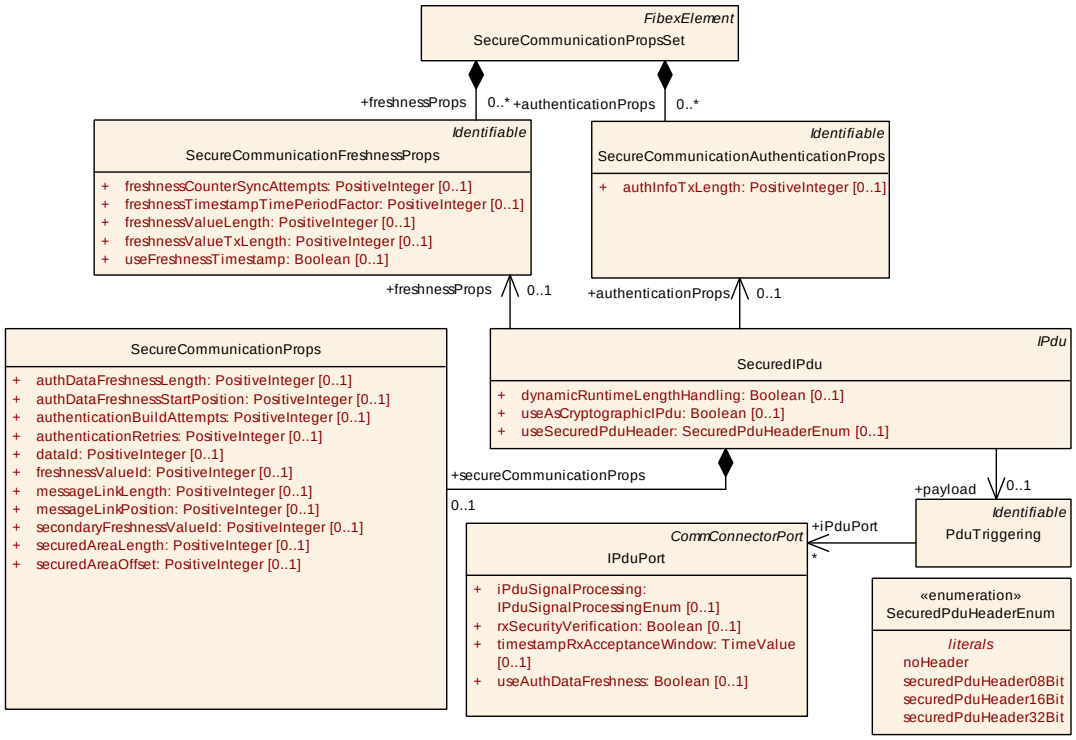

Figure 7.16: Input for for signal based SecOC configuration

7.3.2.4 (D)TLS

A (D)TLS secure channel may provide authenticity, integrity and confidentiality which may be used on combination with SOME/IP and DDS network binding.

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

- TLS_PSK_WITH_NULL_SHA256 for authentic communication (see [29])
- TLS_PSK_WITH_AES_128_GCM_SHA256 for confidential communication (see [29])

7.3.2.4.1 SOME/IP Network binding

[SWS_CM_90103] TLS secure channel for ServiceInterface content using reliable transport

Upstream requirements: RS_CM_00801

⌈A TLS secure channel shall be created and used if a TlsSecureComProps instance is referenced in the role secureComPropsForTcp by a ServiceInstanceToMachineMapping. All content of the ServiceInterface that is referenced by the AdaptivePlatformServiceInstance that in turn is referenced by the ServiceInstanceToMachineMapping that is configured for transmission over “tcp” in the ServiceInterfaceDeployment is selected for transmission over the TLS secured channel.⌋

[SWS_CM_90104] DTLS secure channel for ServiceInterface content using unreliable transport

Upstream requirements: RS_CM_00801

⌈A DTLS secure channel shall be created and used if a TlsSecureComProps instance is referenced in the role secureComPropsForUdp by a ServiceInstanceToMachineMapping. All content of the ServiceInterface that is referenced by the AdaptivePlatformServiceInstance that in turn is referenced by the ServiceInstanceToMachineMapping that is configured for transmission over “udp” in the ServiceInterfaceDeployment is selected for transmission over the TLS secured channel.⌋

[SWS_CM_90121] TLS server role of a Skeleton

Upstream requirements: RS_CM_00801

⌈The TLS secure channel shall be associated with the respective Skeleton and the implementation shall act as a TLS server, if the AdaptivePlatformServiceInstance referenced in

- [SWS_CM_90103]
- [SWS_CM_90104]

is a ProvidedApServiceInstance.⌋

According to the constraints [constr_3485] and [constr_3486] a Proxy and Skeleton cannot be bound to the identical local endpoint (IP address and port). Hence, a local

endpoint can either act as a TLS client or as a TLS server exclusively. However, if multiple `Proxys` are bound to the same endpoint, their common channel shall be shared in the middleware. Likewise, if multiple `Skeletons` are bound to the same endpoint, their common channel shall be shared in the middleware.

[SWS_CM_90119] Behavior of a creating ServiceProxy over TLS or DTLS

Upstream requirements: RS_CM_00804

⌈The instantiation according to [SWS_CM_00131] shall trigger the asynchronous handshake.⌋

[SWS_CM_90111] Behavior of a ServiceProxy over TLS before successful completion of the handshake

Upstream requirements: RS_CM_00804

⌈The communication channel is ready as soon as the TLS handshake is completed.

Therefore, the future returned by the following methods shall only be satisfied after the handshake has finished and once the communication was successful:

- the `operator()` of the respective `Method` class (see [SWS_CM_00196])
- the `Set()` method of the respective `Field` class (see [SWS_CM_00113])
- the `Get()` method of the respective `Field` class (see [SWS_CM_00112])

If the handshake fails, the error code `ComErrc::kPeerIsUnreachable` shall be returned in the `ara::core::Future` of the respective methods (operator(), `Set()`, `Get()`). The error shall be logged.⌋

[SWS_CM_90112] Behavior of a ServiceProxy over DTLS before successful completion of the handshake

Upstream requirements: RS_CM_00804

⌈The communication channel is ready as soon as the DTLS handshake is completed. Before completion the middleware shall drop all requests as if the remote peer is unreachable.⌋

The rationale for choosing different behavior in [SWS_CM_90111] and [SWS_CM_90112] is to reflect the nature of the underlying transport. E.g. plain UDP would also silently discard packets that cannot be sent, where TCP would report an error.

[SWS_CM_90113] Behavior of a ServiceSkeleton over TLS before successful completion of the handshake

Upstream requirements: RS_CM_00804

⌈The communication channel is ready as soon as the TLS handshake is completed. Therefore, [SWS_CM_10287] and [SWS_CM_10319] shall be extended to checking whether the TLS handshake did successfully finish.

Therefore, as if the proxy was not connected, the invocation of the following methods shall not result in sending any data:

- the `Send()` method of the respective `Event` class (see [SWS_CM_00162])
- the `Send()` method of the respective `Trigger` class (see [SWS_CM_00721])
- the `Update()` method of the respective `Field` class (see [SWS_CM_00119])

⌋

[SWS_CM_90114] Behavior of a ServiceSkeleton over DTLS before successful completion of the handshake

Upstream requirements: RS_CM_00804

⌈The communication channel is ready as soon as the TLS handshake is completed. Therefore, [SWS_CM_10287] and [SWS_CM_10319] shall be extended to checking whether the TLS handshake did successfully finish.

Therefore, as if the proxy was not connected, the invocation of the following methods shall not result in sending any data:

- the `Send()` method of the respective `Event` class (see [SWS_CM_00162])
- the `Send()` method of the respective `Trigger` class (see [SWS_CM_00721])
- the `Update()` method of the respective `Field` class (see [SWS_CM_00119])

⌋

7.3.2.4.2 DDS Network Binding (secure transports)

DDS is built upon the Real-Time Publish-Subscribe (RTPS) wire protocol, which allows different implementations of the standard to interoperate at the wire level. The DDS-RTPS specification [21] defines the wire protocol using a Model Driven Architecture; i.e., in terms of a Platform-Independent Model (PIM), which can be mapped to Platform Specific Models (PSM) targeting different transport protocols. In particular, [21] defines

a UDP PSM, and different DDS vendors have implemented TCP PSMs[6], and Shared Memory PSMs for Inter-Process Communication (IPC).

For consistency with the secure channel modeling and secure communication mechanisms specified in 7.3.2.4.1, this section defines support for communication over the following security protocols:

- DTLS, for secure communication over UDP.
- TLS, for secure communication over TCP.
- IPSec, for secure communication over IP.

[SWS_CM_90203] TLS secure channel for methods using reliable transport

Upstream requirements: RS_CM_00801

⌈A TLS secure channel shall be created and used if:

- a `TlsSecureComProps` instance is referenced in the role `secureComProps-ForTcp` by a `ServiceInstanceToMachineMapping` and a `Method` of the `AdaptivePlatformServiceInstance` is selected for transmission over the secure channel by the `ServiceInterfaceElementSecureComConfig` and this `Method` is configured for transmission over “tcp” by `transportProtocol` in the associated `DdsServiceInterfaceDeployment`.

The DataReaders and DataWriters associated with the `Method` shall be configured to operate over TLS.⌋

[SWS_CM_90204] DTLS secure channel for methods using unreliable transport

Upstream requirements: RS_CM_00801

⌈A DTLS secure channel shall be created and used if:

- a `TlsSecureComProps` instance is referenced in the role `secureComProps-ForUdp` by a `ServiceInstanceToMachineMapping` and a `Method` of the `AdaptivePlatformServiceInstance` is selected for transmission over the secured channel by the `ServiceInterfaceElementSecureComConfig` and this `Method` is configured for transmission over “udp” by `transportProtocol` in the associated `DdsServiceInterfaceDeployment`.

The DataReaders and DataWriters associated with the `Method` shall be configured to operate over DTLS.⌋

[SWS_CM_90205] TLS secure channel for events using reliable transport

Upstream requirements: RS_CM_00801

⌈A TLS secure channel shall be created and used if:

[6] A standard TCP PSM for DDS-RTPS is under development, the RFP document is publicly available at the Object Management Group website: `https://www.omg.org/cgi-bin/doc.cgi?mars/2017-9-24`.

- A TlsSecureComProps instance is referenced in the role secureComProps-ForTcp by a ServiceInstanceToMachineMapping and an event of the AdaptivePlatformServiceInstance is selected for transmission over the secured channel by the ServiceInterfaceElementSecureComConfig and this event is configured for transmission over “tcp” by transportProtocol in the associated DdsEventDeployment.

The DataReaders and DataWriters associated with the event shall be configured to operate over TLS.⌋

[SWS_CM_90206] DTLS secure channel for events using unreliable transport

Upstream requirements: RS_CM_00801

⌈A DTLS secure channel shall be created and used if:

- a TlsSecureComProps instance is referenced in the role secureComProps-ForUdp by a ServiceInstanceToMachineMapping and an event of the AdaptivePlatformServiceInstance is selected for transmission over the secured channel by the ServiceInterfaceElementSecureComConfig and this event is configured for transmission over “udp” by transportProtocol in the associated DdsEventDeployment.

The DataReaders and DataWriters associated with the event shall be configured to operate over DTLS.⌋

[SWS_CM_90207] TLS secure channel for fields

Upstream requirements: RS_CM_00801

⌈The requirements [SWS_CM_90203], [SWS_CM_90204], [SWS_CM_90205] and [SWS_CM_90206] apply to fields in the same manner, since fields are a composition of methods and events.⌋

[SWS_CM_90209] IPsec secure channel between communication nodes and Transport of Service communication over an IPsec security association

Upstream requirements: RS_CM_00801

⌈An IPsec secure channel shall be created and used according to the requirements and constraints specified in [SWS_CM_90117] and [SWS_CM_90118].⌋

7.3.2.5 IPsec

IPsec provides cryptographic protection for IP datagrams in IPv4 and IPv6 network packets.

[SWS_CM_90117] IPsec secure channel between communication nodes

Upstream requirements: RS_CM_00801

⌈An IPsec secure channel shall be created and used if an AdaptivePlatformServiceInstance is mapped by ServiceInstanceToMachineMapping to an EthernetCommunicationConnector that points with the unicastNetworkEndpoint to a NetworkEndpoint that aggregates an IPSecConfig.

The IPSecRules in the IPSecConfig define security associations between the NetworkEndpoint that aggregates this IPSecConfig and remote nodes that are defined by the referenced remoteIpAddress.⌋

[SWS_CM_90118] Transport of Service communication over an IPsec security association

Upstream requirements: RS_CM_00801

⌈If a communication connection is established between a Service Provider and Service Requester and the configured transport layer connection matches the defined security association then the IP packets exchanged between the Service Provider and Service Requester will be protected by IPsec.

In other words it means that if the IPsec security association defined by

- the local Address (IP Address defined by the networkEndpointAddress, Port and Protocol defined by localPortRangeStart and localPortRangeEnd)
- the remote Address (IP Address defined by the remoteIpAddress, Port and Protocol defined by remotePortRangeStart or remotePortRangeEnd)

equals the settings defined by

- the ServiceInstanceToMachineMapping for the ProvidedApServiceInstance and
- the ServiceInstanceToMachineMapping for the RequiredApServiceInstance and
- this network connection is established

then the IP packets between the two nodes will be protected according to the configuration that is also defined in the IPSecRule.⌋

7.3.2.6 MACsec

MACsec provides cryptographic protection for MAC frames.

[SWS_CM_99040] MACsec secure channel between communication nodes and MACsec security association

Upstream requirements: FO_RS_MACsec_00001, FO_RS_MACsec_00006

⌈A MACsec secure channel and secure association shall be created and used according to the requirements and constraints specified in [SWS_CM_00154] and [SWS_CM_00155].⌋

[SWS_CM_00154] MACsec secure channel between communication nodes ⌈A MACsec secure channel shall be created and used if an `AdaptivePlatformServiceInstance` is mapped by a `ServiceInstanceToMachineMapping` to a `CommunicationConnector` that references a `EthernetCommunicationController` that in turn aggregates a `CouplingPort` with aggrageted `MacSecProps`. `MacSecProps` define the settings for the MACsec Key Agreement that is responsible for the peer discovery and key negotiation to secure the Ethernet link.⌋

[SWS_CM_00155] Communication over a MACsec security association

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

The frames exchanged between the MACsec nodes (called `KeyServer` and `Peer` in MACsec secure channel communication) shall be protected if the configured MAC layer address matches an established secure channel and security association. The frames to be protected and validated shall match:

- The Ethertype and, if present, VLAN of the frame are not included in the corresponding `bypassEtherType` or `bypassVlan` defined on the `MacSecGlobalKayProps` of the corresponding `CouplingPort`.
- The source MAC Address of the frame matches the MAC Address defined by the `destinationMacAddress` defined by `MacSecLocalKayProps` and `MacSecKayParticipant` which is part of the `EthernetCluster`.

where each node, with the previous MAC Addresses, is a `MacSecKayParticipant` included in the `EthernetCluster`, then the MAC packets between the two nodes will be protected according to the configuration that is also defined in `MacSecGlobalKayProps`.⌋

7.4 Safety

In the following chapter the behavior according to the meta-model of safety communication shall be described.

[SWS_CM_00089] Serialization of calls to transport fault condition handlers for Events, Triggers, Field Notifiers, Method requests and Field Get and Set requests

Upstream requirements: RS_CM_00211, RS_CM_00212, RS_CM_00213, RS_CM_00214, RS_CM_00215, RS_CM_00216, RS_CM_00217, RS_CM_00218

⌈The Communication Management shall serialize calls to the registered TransportFaultConditionHandler function for:

- Individual Events of individual Service Instances ([SWS_CM_00099], [SWS_CM_00100] and [SWS_CM_00105])
- Individual Triggers of individual Service Instances ([SWS_CM_00106], [SWS_CM_00107] and [SWS_CM_00108])
- Individual Notifiers of individual Service Instances ([SWS_CM_00109], [SWS_CM_00110] and [SWS_CM_00126])
- All Method requests of individual Service Instances ([SWS_CM_00093], [SWS_CM_00094] and [SWS_CM_00095])
- All Field Get and Set requests of individual Service Instances ([SWS_CM_00096], [SWS_CM_00097] and [SWS_CM_00098])

⌋

[SWS_CM_00090] Client-side reporting of transport fault conditions for Methods and Field Getters and Setters

Upstream requirements: RS_CM_00205

⌈Method, Field `Get` and `Set` methods will report any transport fault condition by reporting `ComErrc::kNetworkBindingFailure` through their resulting `ara::core::Future`s.⌋

7.4.1 End-to-end communication protection for SOMEIP

7.4.1.1 Events

This section specifies the integration of E2E communication protection in `ara::com` for the processing of `Event`s.

[SWS_CM_90402] E2E event protection properties and profile configuration

Upstream requirements: RS_E2E_08540

⌈An E2E-protected `Event` shall have its options configured in `End2EndEventProtectionProps` and `E2EProfileConfiguration`.⌋

[SWS_CM_90433] Requirements of E2E_protect and E2E_check

Upstream requirements: RS_E2E_08540, RS_CM_00223

⌈The E2E functions mentioned in this section using the names `E2E_protect` and `E2E_check` shall meet the requirements on E2E protection as defined in [9] and comply with the E2E protection protocol specification of [7] (especially [PRS_E2E_00323]).⌋

For each specific `Event` class belonging to a specific `ServiceProxy`/`ServiceSkeleton` class the E2E `dataID` - based on, e.g., a combination of Service ID, Service Instance ID and Event ID - is available.

7.4.1.1.1 Limitations

The specified E2E communication protection for `event`s is limited.

- `EndToEndTransformationComSpecProps` are not supported.

General limitations regarding E2E protection and the detectable failure modes are described in [7].

The values of the following E2E parameters are defined as fixed by the standard and shall not be changed. See [PRS_E2E_00324] of [7]:

- `counterOffset`
- `crcOffset`
- `dataIdNibbleOffset`

The value of following E2E parameters shall be set to the default values specified by [PRS_E2E_00324] of [7]:

- `offset`

The value of `dataIdMode` for `Event`s and the notifier of `Field`s shall be set according to the `dataIdMode` of the `E2EProfileConfiguration` which is referenced (in role `e2eProfileConfiguration`) by the `AdaptivePlatformServiceInstance`.`e2eEventProtectionProps` which reference (in role `event`) the `ServiceEventDeployment` of the particular `Event` or the `Field` notifier.

7.4.1.1.2 Publisher

[SWS_CM_00046] E2E protection of events in Send

Upstream requirements: RS_CM_00223, RS_E2E_08540

⌈For E2E-protected `Event`s, E2E protection shall be performed within the context of `Send()` / `Send()`.⌋

Figure 7.17 shows an overview of the interaction of components involved during the E2E protection at the publisher side.

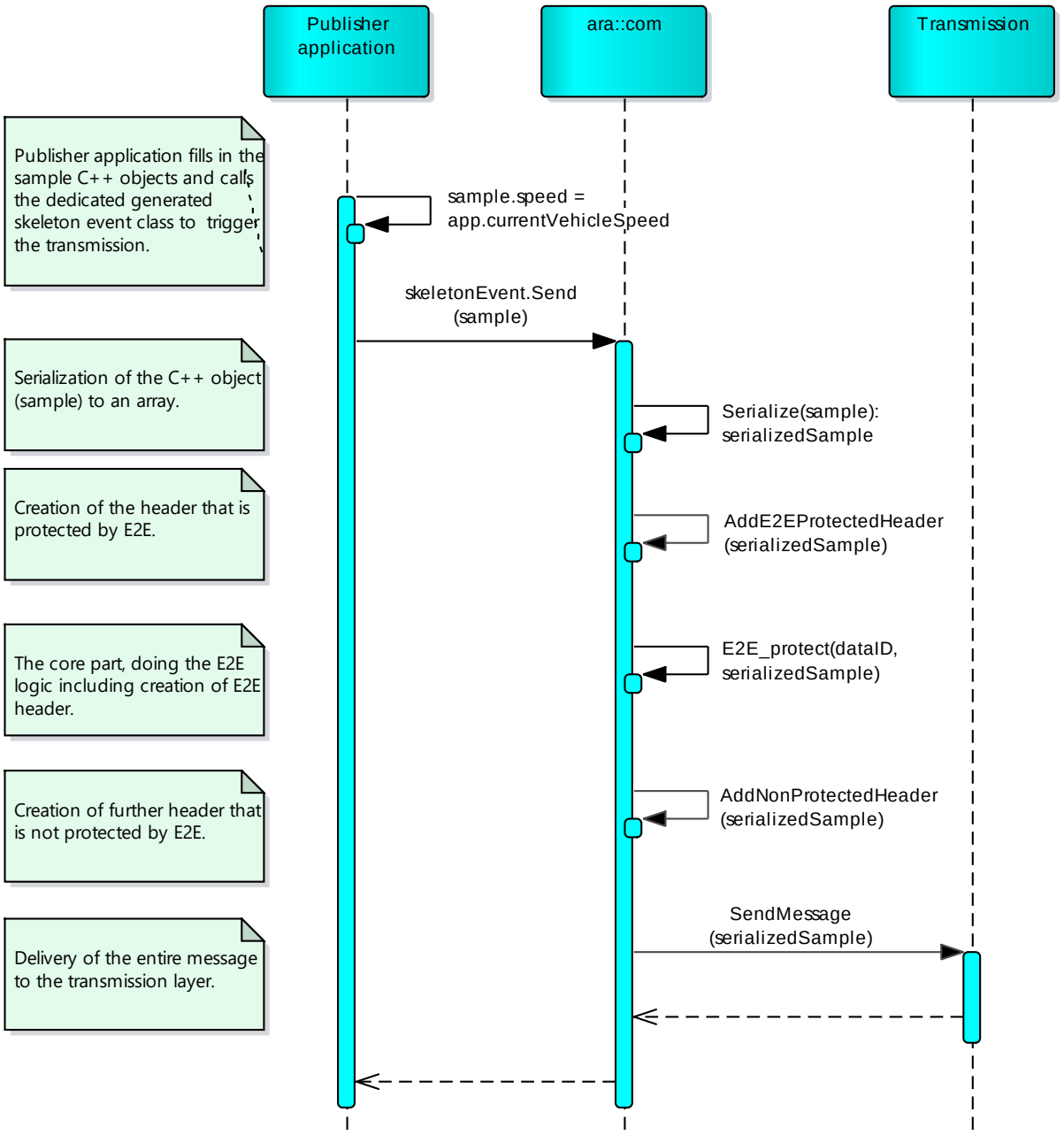

Figure 7.17: E2E Publisher

[SWS_CM_90430] E2E-protected events sample serialization

Upstream requirements: RS_CM_00223, RS_E2E_08540

⌈For E2E-protected `Events`, `Send()` / `Send()` shall serialize the sample and potentially add a protocol header according to the rules of the respective network binding

(e.g., according to [SWS_CM_10291] in case of SOME/IP network binding), resulting in serialized data.⌋

From E2E protection perspective this serialized data include both a non-protected part as well as the part to be protected (see [PRS_E2E_UC_00239] and [PRS_E2E_USE_-00741]).

[SWS_CM_90401] E2E_protect for event serialized data

Upstream requirements: RS_E2E_08540

⌈For E2E-protected `Event`s, `E2E_protect` shall be invoked on the to be protected serialized data (passed as argument `serializedData` to `E2E_protect`) according to [PRS_E2E_00323].⌋

[SWS_CM_90403] Argument dataID in E2E_protect for events

Upstream requirements: RS_E2E_08540

⌈For E2E-protected `Event`s, the `End2EndEventProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_protect`.⌋

[SWS_CM_90404] E2E protection header for events

Upstream requirements: RS_E2E_08540

⌈For E2E-protected `Event`s, in case of SOME/IP serialization the E2E protection header shall be added to the message. If the protocol specification of the respective network binding imposes restrictions on the placement of the E2E protection header (e.g., [PRS_SOMEIP_00941] in case of SOME/IP network binding), then these restrictions shall be honored.⌋

7.4.1.1.3 Subscriber - GetNewSamples

[SWS_CM_90406] E2E checking shall be done in GetNewSamples for events

Upstream requirements: RS_CM_00223, RS_E2E_08540

⌈For E2E-protected `Event`s, E2E checking shall be performed within the context of `GetNewSamples()`.⌋

Figure 7.18 shows an overview of the interaction of components involved during the E2E checking at the subscriber side.

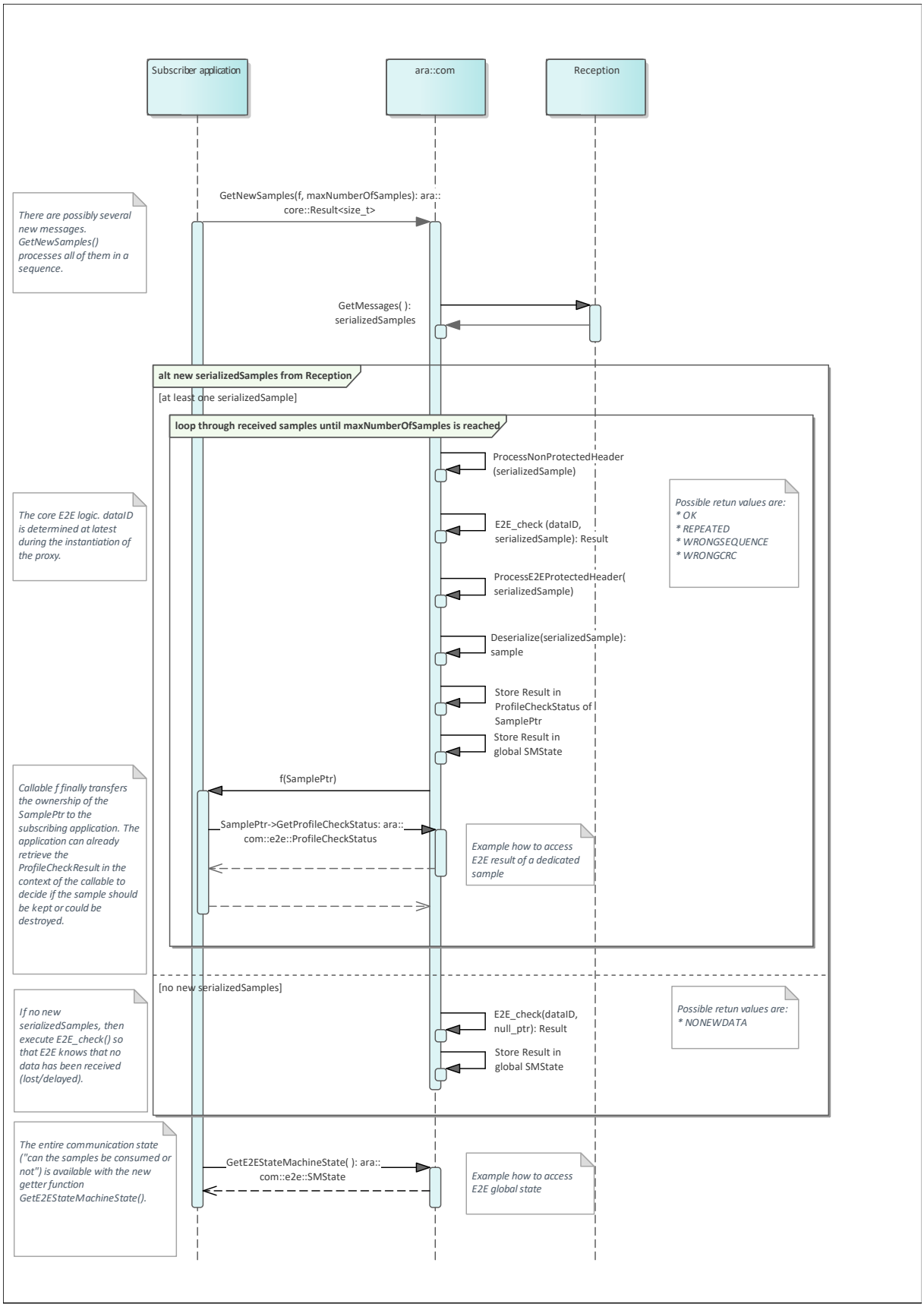

Figure 7.18: E2E Subscriber

[SWS_CM_90407] GetNewSamples shall get all the serialized data that has not yet been fetched

Upstream requirements: RS_CM_00224, RS_E2E_08540

⌈For E2E-protected `Event`s, `GetNewSamples()` shall first get the collection of all serialized data that have not been fetched during the last call of this `GetNewSamples()` function.⌋

From E2E protection perspective this serialized data include both a non-protected part as well as the part to be protected (see [PRS_E2E_UC_00239] and [PRS_E2E_USE_-00741]).

7.4.1.1.3.1 Case 1 - there are one or more serialized samples

For E2E-protected `Event`s, in case serialized data for one or more samples are received, then for each sample, the following steps are to be done:

[SWS_CM_90408] Processing the non-E2E-protected header of E2E-protected sample

Upstream requirements: RS_CM_00224, RS_E2E_08540

⌈For the given E2E-protected sample, `GetNewSamples()` shall process the non-E2E protected header (if any) of the sample's serialized data.⌋

[SWS_CM_90410] E2E_check for event serialized data

Upstream requirements: RS_E2E_08540

⌈For the given E2E-protected sample, `E2E_check` shall be invoked on the protected serialized data (passed as argument `serializedData` to `E2E_check`) according to [RS_E2E_08540] and [PRS_E2E_00323].⌋

[SWS_CM_00045] Argument dataID in E2E_check for event with serialized sample

Upstream requirements: RS_E2E_08540

⌈For the given E2E-protected sample, the `End2EndEventProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_check`.⌋

[SWS_CM_90411] E2E_check for Events provides Result with `ara::com::e2e::SMState` and `ara::com::e2e::ProfileCheckStatus`

Upstream requirements: RS_E2E_08540, RS_E2E_08534

⌈In return, for the given E2E-protected sample, `E2E_check` shall provide a `Result` (`e2eResult` according to [PRS_E2E_00322] of [7]) containing the elements `ara::com::e2e::SMState` (`e2eState` according to [PRS_E2E_00322] of [7] mapped to

the ara::com::e2e::SMState enum literal according to [SWS_CM_12022]) and ara::com::e2e::ProfileCheckStatus (e2eStatus according to [PRS_E2E_-00322] of [7] mapped to the ara::com::e2e::ProfileCheckStatus enum literal according to [SWS_CM_12021]).⌋

[SWS_CM_00044] E2E Protection header removal from serialized data

Upstream requirements: RS_E2E_08540

⌈For the given E2E-protected sample, the E2E protection header shall be removed from the serialized data.⌋

[SWS_CM_90412] E2E-protected sample deserialization

Upstream requirements: RS_CM_00224, RS_E2E_08540

⌈For the given E2E-protected sample, GetNewSamples() shall de-serialize the resulting serialized data according to the rules of the respective network binding (e.g., according to [SWS_CM_10294] in case of SOME/IP network binding), resulting in the deserialized sample.⌋

[SWS_CM_90413] GetNewSamples shall update ara::com::e2e::ProfileCheckStatus in the ara::com::SamplePtr and ara::com::e2e::SMState in the Event class

Upstream requirements: RS_CM_00224, RS_E2E_08540, RS_E2E_08534

⌈For the given E2E-protected sample, GetNewSamples() shall store the ara::com::e2e::ProfileCheckStatus in the ara::com::SamplePtr and shall update/overwrite the global ara::com::e2e::SMState within its specific Event class of the specific E2E-protected Event.⌋

7.4.1.1.3.2 Case 2 - there are no serialized samples

For E2E-protected Events, in case no serialized data are received, the steps are simpler and E2E protection works as timeout detection.

[SWS_CM_90415] E2E_check invoked on a null sample

Upstream requirements: RS_E2E_08540

⌈E2E_check shall be invoked on a null sample (i.e., a null pointer shall be passed as argument serializedData to E2E_check) according to [RS_E2E_08540] and [PRS_E2E_00323].⌋

[SWS_CM_00043] Argument dataID in E2E_check for events without serialized sample

Upstream requirements: RS_E2E_08540

⌈The `End2EndEventProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_check`.⌋

[SWS_CM_90416] E2E_check Result on a null sample

Upstream requirements: RS_E2E_08540, RS_E2E_08534

⌈In return, for the given null sample, `E2E_check` shall provide a `Result` (`e2eResult` according to [PRS_E2E_00322] of [7]) containing the elements `ara::com::e2e::SMState` (`e2eState` according to [PRS_E2E_00322] of [7] mapped to the `ara::com::e2e::SMState` enum literal according to [SWS_CM_12022]) and `ara::com::e2e::ProfileCheckStatus` (`e2eStatus` according to [PRS_E2E_00322] of [7] mapped to the `ara::com::e2e::ProfileCheckStatus` enum literal according to [SWS_CM_12021]).⌋

[SWS_CM_90417] GetNewSamples shall update the `ara::com::e2e::SMState` of specific event class

Upstream requirements: RS_CM_00224, RS_E2E_08540, RS_E2E_08534

⌈`GetNewSamples()` shall update/overwrite the global `ara::com::e2e::SMState` within its specific `Event` class of the specific E2E-protected `Event`.⌋

7.4.1.1.4 Subscriber - Callable f

The user provided `Callable f` is invoked for each received sample. The `Callable f` is called with the `ara::com::SamplePtr` of the corresponding sample as parameter. The `ara::com::SamplePtr` contains the de-serialized sample including the `ara::com::e2e::ProfileCheckStatus`.

7.4.1.1.5 Subscriber - Access to E2E information

[SWS_CM_00042] `GetProfileCheckStatus()` method of `ara::com::SamplePtr`

Upstream requirements: RS_CM_00224, RS_E2E_08540

⌈Each `ara::com::SamplePtr` shall provide a `GetProfileCheckStatus()` method to access the `ara::com::e2e::ProfileCheckStatus` of each sample.⌋

[SWS_CM_10475] GetE2EStateMachineState method for Events

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈A `GetE2EStateMachineState` method shall be provided for each `Event` to provide access to the global `ara::com::e2e::SMState` of the `Event`(see [SWS_CM_11554]).⌋

7.4.1.2 Methods

This section specifies the integration of E2E communication protection in `ara::com` for the processing of `Methods`s. This includes E2E communication protection for a `Method`'s request as well as E2E communication protection for any kind of `Method`'s response (i.e., normal or error response).

[SWS_CM_10460] Options of E2E Protection for Methods

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈An E2E-protected `Method` shall have its options configured in `End2EndMethodProtectionProps` and `E2EProfileConfiguration`.⌋

[SWS_CM_90485] E2E Protection for Methods shall comply E2E Protection protocol specification

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈The E2E functions mentioned in this section using the name `E2E_protect` and `E2E_check` shall meet the requirements on E2E protection as defined in [9] and comply with the E2E protection protocol specification of [7] (especially [PRS_E2E_00828]).⌋

For each specific `Method` class ([SWS_CM_00196]) belonging to a specific `ServiceProxy` class and for each provided method (see [SWS_CM_00191]) belonging to a specific `ServiceSkeleton` class the E2E `dataID` - based on, e.g., a combination of Service ID, Service Instance ID and Method ID - is available.

Within the scope of this section a failed E2E check is an invocation of `E2E_check` returning an `e2eStatus` of either `REPEATED`, `WRONGSEQUENCE` or `NONEWDATA`. A successful E2E check is an invocation of `E2E_check` returning an `e2eStatus` different from `REPEATED`, `WRONGSEQUENCE`, and `NONEWDATA`.

7.4.1.2.1 Limitations

The specified E2E communication protection for `Method`s is limited.

- The `ara::com::MethodCallProcessingMode==kEvent` (concurrent threads) is not supported for E2E protected methods.
- `EndToEndTransformationComSpecProps` are not supported.

General limitations regarding E2E protection and the detectable failure modes are described in [7].

The values of the following E2E parameters are defined as fixed by the standard and shall not be changed. See [PRS_E2E_00324] of [7]:

- `counterOffset`
- `crcOffset`
- `dataIdNibbleOffset`

The value of following E2E parameters shall be set to the default values specified by [PRS_E2E_00324] of [7]:

- `offset`

The value of `dataIdMode` for `Method`s and the getters and setters of `Field`s shall be set according to the dataIdMode of the `E2EProfileConfiguration` which is referenced (in role `e2eProfileConfiguration`) by the `AdaptivePlatformServiceInstance.e2eMethodProtectionProps` which reference (in role `method`) the `ServiceMethodDeployment` of the particular `Method` or the `Field` getter/setter.

7.4.1.2.2 E2E protection of the service method request (Client)

[SWS_CM_10462] E2E-protected Methods Request Message Protection

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method`s, E2E protection of the request message shall be performed within the context of the `operator()` of the `Method` class (see [SWS_CM_00196]) of the respective service method.⌋

Figure 7.19 shows an overview of the interaction of components involved during the E2E protection of the `Method` request at the client side.

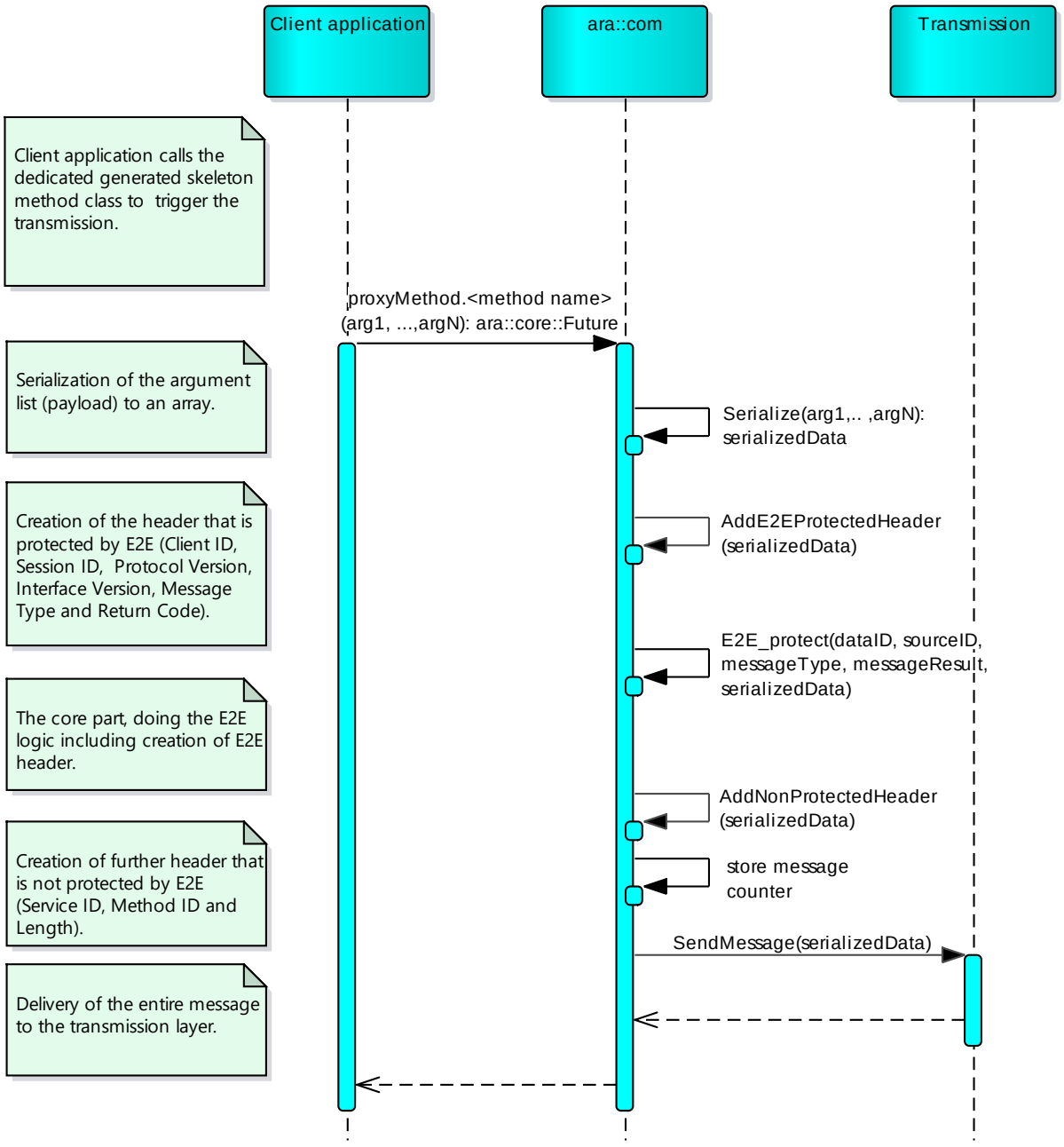

Figure 7.19: Interaction of components during E2E protection of the `Method` request at the client side

7.4.1.2.2.1 Serializing the payload

[SWS_CM_00041] E2E-protected Methods Arguments Serialization

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests, `operator()` shall serialize the `Method`'s `in` and `inout` arguments and potentially add a protocol header according to the rules of the respective network binding (e.g., according to [SWS_CM_10301] in case of SOME/IP network binding), resulting in the serialized data.⌋

From E2E protection perspective this serialized data include both a non-protected part as well as the part to be protected (see [PRS_E2E_UC_00239] and [PRS_E2E_USE_-00741]).

7.4.1.2.2.2 E2E protection of the payload

[SWS_CM_90479] E2E-protected Methods Serialized Data Protection

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests, `E2E_protect` shall be invoked on the to be protected serialized data (passed as argument `serializedData` to `E2E_protect`) according to [RS_E2E_08541], [PRS_E2E_00323], and [PRS_E2E_00828].⌋

[SWS_CM_10463] E2E-protected Method Requests dataID Argument

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests, the `End2EndMethodProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_protect`.⌋

[SWS_CM_90486] Argument sourceID for E2E_protect

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests using profiles P04m, P07m, P08m, or P44m, the `End2EndMethodProtectionProps.sourceId` shall be passed as argument `sourceID` to `E2E_protect`.⌋

[SWS_CM_90487] Argument messageType for E2E_protect

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests using profiles P04m, P07m, P08m, or P44m, `STD_MESSAGETYPE_REQUEST (0)` shall be passed as argument `messageType` to `E2E_protect`.⌋

[SWS_CM_90488] Argument messageResult for E2E_protect

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests using profiles P04m, P07m, P08m, or P44m, `STD_MESSAGERESULT_OK` (`0`) shall be passed as argument `messageResult` to `E2E_protect`.⌋

[SWS_CM_10464] E2E protection header according to the network binding in the method request

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests, the E2E protection header shall be added to the message. If the protocol specification of the respective network binding imposes restrictions on the placement of the E2E protection header (e.g., [PRS_SOMEIP_00941] in case of SOME/IP network binding), then these restrictions shall be honored.⌋

7.4.1.2.3 E2E checking the service method request (Server)

[SWS_CM_10466] E2E checking of the method request in ServiceSkeleton (message reception)

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests, E2E checking shall be performed within the context of the message reception within the `ServiceSkeleton` if the `ara::com::MethodCallProcessingMode`==`kEventSingleThread`.⌋

[SWS_CM_10468] E2E checking of the method request in ServiceSkeleton (ProcessNextMethodCall)

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests, E2E checking shall be performed within the context of `ProcessNextMethodCall()` within the `ServiceSkeleton` if the `ara::com::MethodCallProcessingMode` is set to `kPoll`.⌋

Figures 7.20 and 7.21 show an overview of the interaction of components involved during the E2E checking of the `Method` request at the server side.

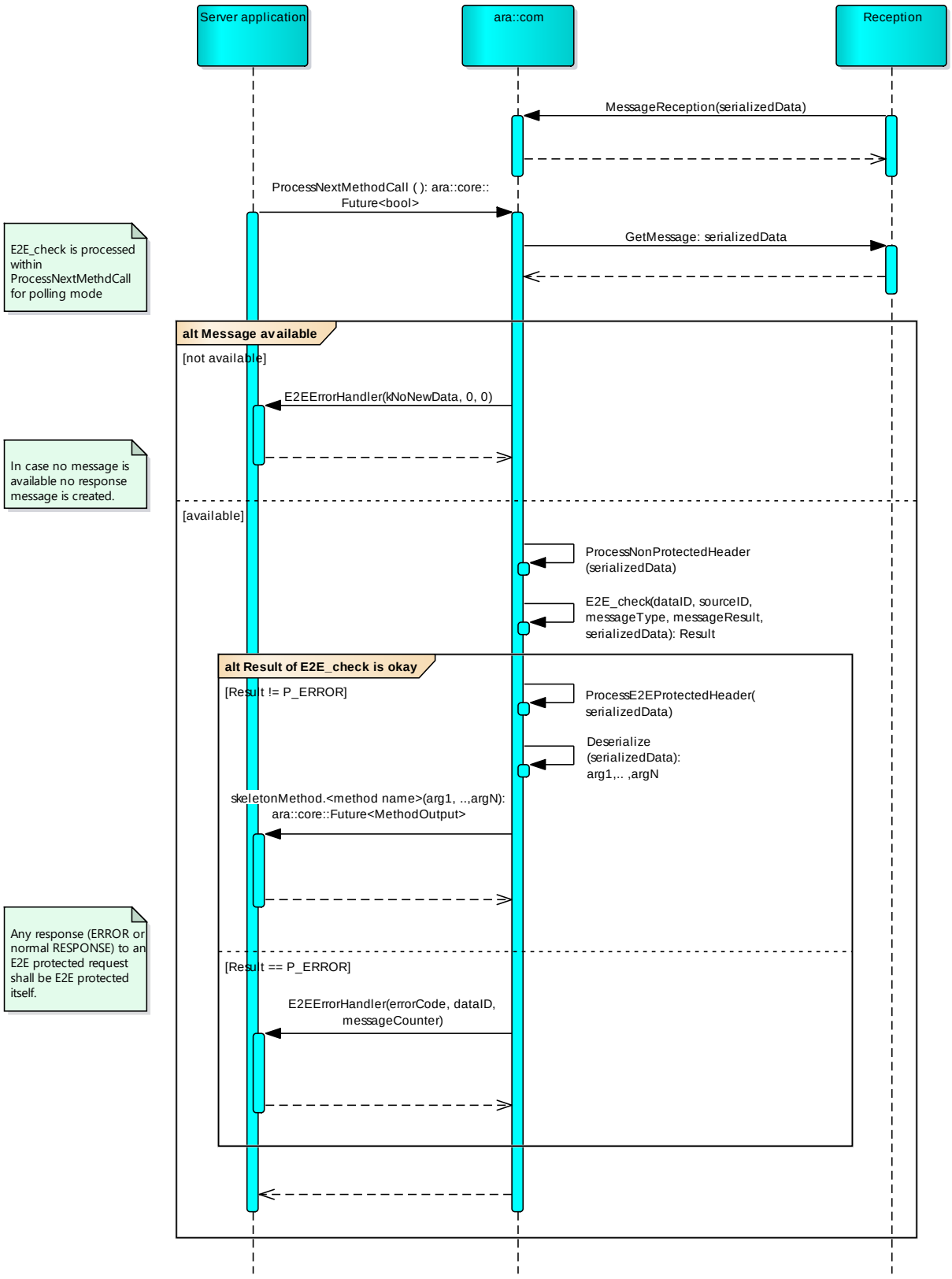

Figure 7.20: Interaction of components during E2E checking of the `Method` request at the server side - polling

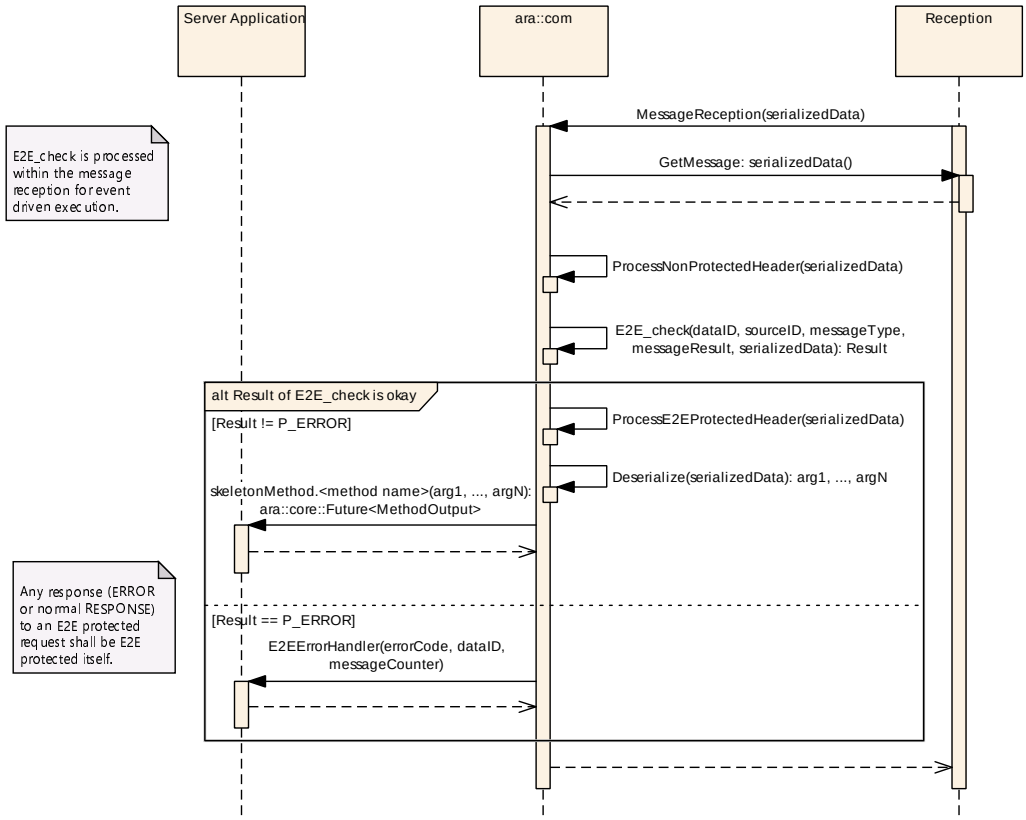

Figure 7.21: Interaction of components during E2E checking of the `Method` request at the server side - event driven

7.4.1.2.3.1 E2E checking of the payload

For E2E-protected `Method` requests, in case serialized data are available the following steps are to be done:

[SWS_CM_00040] Processing the non-E2E-protected header of E2E-protected method request

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected `Method` request, the non-E2E-protected header (if any) of the `Method` request's serialized data shall be processed.⌋

[SWS_CM_90480] Argument serializedData in E2E_check for method requests

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected `Method` request, `E2E_check()` shall be invoked on the protected serialized data (passed as argument `serializedData` to `E2E_check()`) according to [RS_E2E_08541], [PRS_E2E_00323], and [PRS_E2E_00828].⌋

[SWS_CM_00039] Argument dataID in E2E_check for method requests

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected `Method` request, the `End2EndMethodProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_check()`.⌋

[SWS_CM_90489] Argument sourceID in E2E_check for method requests

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests using profiles P04m, P07m, P08m, or P44m, a reference to a variable to store the `End2EndMethodProtectionProps.sourceId` to shall be passed as argument `sourceID` to `E2E_check`. `E2E_check` shall extract the E2E Source ID contained in the E2E protection header into this variable. This extracted `sourceID` shall be stored for later use during E2E protection of response payload (see [SWS_CM_90492]).⌋

[SWS_CM_90490] Argument messageType in E2E_check for method requests

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests using profiles P04m, P07m, P08m, or P44m, `STD_MESSAGETYPE_REQUEST` (`0`) shall be passed as argument `messageType` to `E2E_check`.⌋

[SWS_CM_90491] Argument messageResult E2E_check for method requests

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` requests using profiles P04m, P07m, P08m, or P44m, `STD_MESSAGERESULT_OK` (`0`) shall be passed as argument `messageResult` to `E2E_check`.⌋

[SWS_CM_00038] E2E_check for method request provides Result with SMState and `ara::com::e2e::ComE2EErrc`

Upstream requirements: RS_E2E_08541, RS_E2E_08534

⌈In return, for the given E2E-protected `Method` request, `E2E_check` shall provide a `Result` (`e2eResult` according to [PRS_E2E_00322] of [7]) containing the elements `ara::com::e2e::SMState` (`e2eState` according to [PRS_E2E_00322] of [7]) and `ara::com::e2e::ComE2EErrc` (`e2eStatus` according to [PRS_E2E_00322] of [7] mapped to the `ara::com::e2e::ComE2EErrc` enum literal according to [SWS_CM_12026]).⌋

[SWS_CM_00037] E2E Protection header removal from serialized data for method requests

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected `Method` request, the E2E protection header shall be removed from the serialized data.⌋

7.4.1.2.3.2 Deserializing the payload

In case the call to `E2E_check` (according to [SWS_CM_00040]) indicated a successful E2E check of the request message further processing of the request message shall take place.

[SWS_CM_00036] Deserialization of the data according to the network binding for method request

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected `Method` request, the resulting serialized data shall be de-serialized according to the rules of the respective network binding (e.g., according to [SWS_CM_10304] in case of SOME/IP network binding), resulting in the de-serialized `in` and `inout` arguments to the `Method` call.⌋

7.4.1.2.3.3 E2E error notification

In case the call to `E2E_check` (according to [SWS_CM_00040]) indicated a failed E2E check of the request message, the server application can get notified via an E2E error handler.

The registration of an application's E2E error handler is static (before runtime). A dynamic registration/de-registration of an application's E2E error handler (like a publisher/subscriber pattern) is neither necessary nor possible.

[SWS_CM_10470] E2E Error Handler - Existence

Upstream requirements: RS_CM_00401, RS_CM_00402

⌈The `ServiceSkeleton` shall provide a virtual `E2EErrorHandler` method with arguments for `errorCode`, `dataID`, and `messageCounter`. This `E2EErrorHandler` function shall have an empty implementation which may be overridden by the actual `ServiceSkeleton` implementation. The `E2EErrorHandler` implementation is not required to be reentrant.

```
virtual void E2EErrorHandler(
    ara::com::e2e::ComE2EErrc errorCode,
    ara::com::e2e::DataID dataID,
    ara::com::e2e::MessageCounter messageCounter
```

```
    )
{
};
```

⌋

Note - Faulty DataID: If the E2E error is a CRC error then some parts of the received message are faulty. If this part is the DataID then the E2E error handler is called with a faulty DataID. Consequently, in case of CRC error the server application can not rely on the DataID received by its error handler.

[SWS_CM_00034] E2E Error Handler - Invocation

Upstream requirements: RS_CM_00401, RS_CM_00402

⌈`E2EErrorHandler` shall be invoked from within a separate thread by the Communication Management software in case `E2E_check` reports an E2E error.⌋

[SWS_CM_10471] E2E Error Handler - Invocation Arguments

Upstream requirements: RS_CM_00401, RS_CM_00402

⌈In case a new request message is available, `E2EErrorHandler` shall be called with the following arguments: `errorCode` shall be set to the `ara::com::e2e::ComE2EErrc` obtained in [SWS_CM_00038], `dataID` shall be set to `End2EndMethodProtectionProps.dataId`, and `messageCounter` shall be set to the E2E counter of the received request message.⌋

[SWS_CM_00047] E2E Error Handler - Invocation Arguments

Upstream requirements: RS_CM_00401, RS_CM_00402

⌈In case no new request message is available, `E2EErrorHandler` shall be called with the following arguments: `errorCode` shall be set to the `kNoNewData`, `dataID` shall be set to `0`, and `messageCounter` shall be set `0`.⌋

7.4.1.2.4 E2E protection of the service method response (Server)

[SWS_CM_90481] E2E protection of method response message performed after the method or E2E error handler execution

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method`s, E2E protection of the response message shall be performed after the execution of the service method (in case of a successful `E2E_check` according to [SWS_CM_90480]) or after the execution of the E2E error handler (in case of a failed E2E check according to [SWS_CM_90480]).⌋

Figure 7.22 shows an overview of the interaction of components involved during the E2E protection of the `Method` response at the server side.

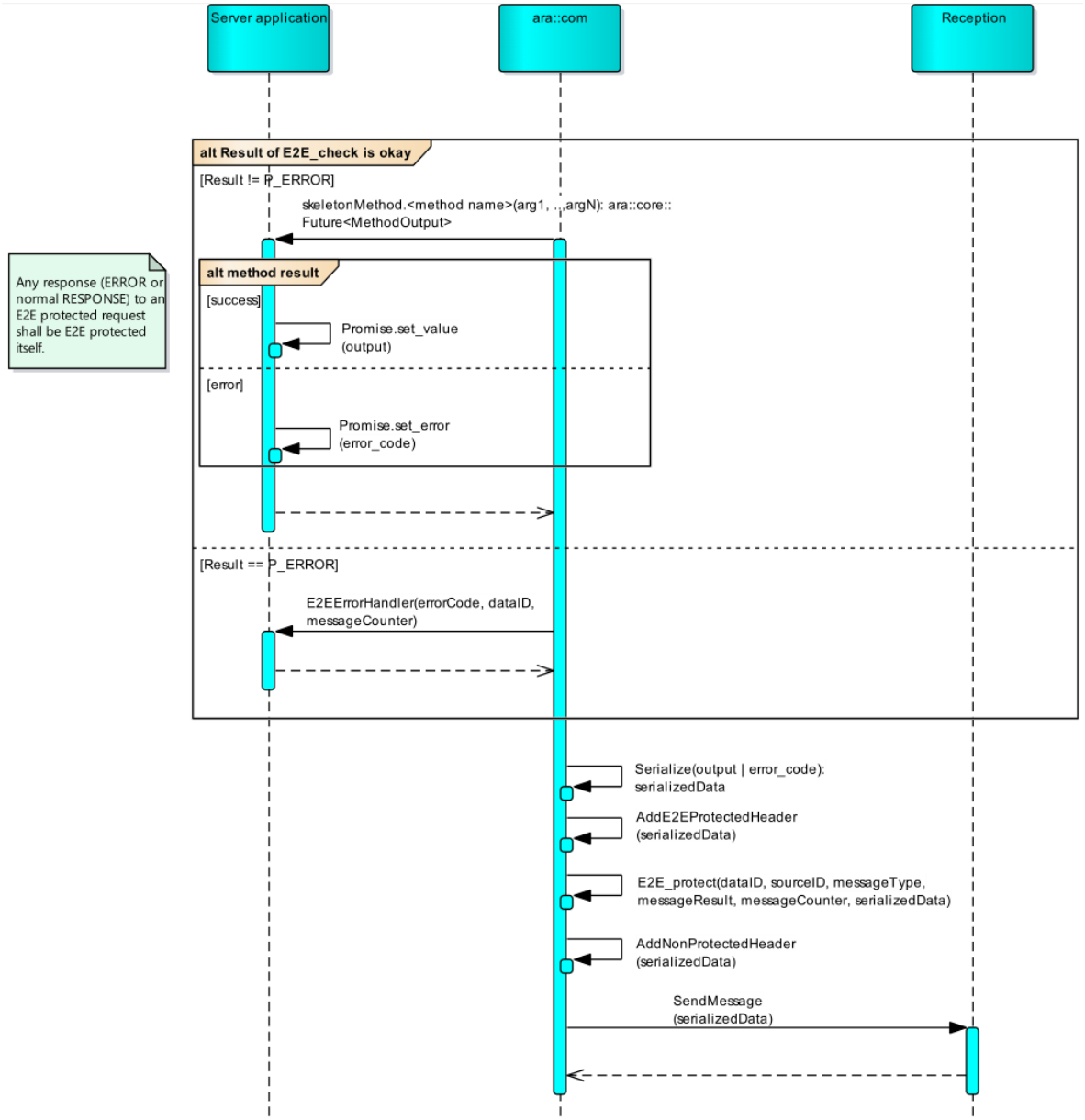

Figure 7.22: Interaction of components during E2E protection of the `Method` response at the server side

7.4.1.2.4.1 Serializing the E2E error response payload

[SWS_CM_10472] E2E Error Response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈In case `E2E_check` (according to [SWS_CM_90480]) reported an E2E error, an error response message according to the used network binding (e.g., [SWS_CM_10312] in case of SOME/IP) shall be sent to the client.⌋

[SWS_CM_00033] Payload of the E2E Error Response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈The payload of this error response message shall contain an `ara::core::ErrorCode` of error domain `ara::com::e2e::E2EErrorDomain`. The value of this `ara::core::ErrorCode` shall be set to the corresponding error value of `E2E_check` according to [SWS_CM_90421]. The serialization of this error code and the potential adding of a protocol header shall take place according to the used network binding (e.g., according to [SWS_CM_10312] and [SWS_CM_10428] in case of SOME/IP).⌋

7.4.1.2.4.2 Serializing the response payload

[SWS_CM_90467] Payload of the Normal or Application Error Response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method`s the `Method` `inout` and `out` arguments or the application error shall be serialized and a protocol header shall be potentially added according to the rules of the respective network binding (e.g., according to [SWS_CM_10312] in case of SOME/IP network binding), resulting in the serialized data.⌋

From E2E communication protection perspective this serialized data include both a non-protected part as well as the part to be protected (see [PRS_E2E_UC_00239] and [PRS_E2E_USE_00741]).

7.4.1.2.4.3 E2E protection of the response payload

[SWS_CM_90468] Argument serializedData in E2E_protect for methods

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` responses, `E2E_protect` shall be invoked on the to be protected serialized data (passed as argument `serializedData` to `E2E_protect`) according to [RS_E2E_08541], [PRS_E2E_00323], and [PRS_E2E_00828].⌋

[SWS_CM_10469] Argument dataId in E2E_protect for methods

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method responses, the End2EndMethodProtectionProps.dataId shall be passed as argument dataID to E2E_protect.⌋

Note: This is the same dataID that has been contained in the corresponding Method request.

[SWS_CM_90492] Argument sourceId in E2E_protect for methods

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method responses using profiles P04m, P07m, P08m, or P44m, the stored sourceID (which has been extracted according to [SWS_CM_90489]) shall be passed as argument sourceID to E2E_protect.⌋

[SWS_CM_90493] Argument messageType in E2E_protect for methods

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method responses using profiles P04m, P07m, P08m, or P44m, STD_MESSAGETYPE_RESPONSE (1) shall be passed as argument messageType to E2E_protect.⌋

[SWS_CM_90494] Argument messageResult STD_MESSAGERESULT_OK in E2E_protect for methods

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method responses using profiles P04m, P07m, P08m, or P44m, in case of a normal response (i.e., neither an application error response message nor an E2E error response message), STD_MESSAGERESULT_OK (0) shall be passed as argument messageResult to E2E_protect.⌋

[SWS_CM_90495] Argument messageResult STD_MESSAGERESULT_ERROR in E2E_protect for methods

Upstream requirements: RS_CM_00401, RS_E2E_08541

⌈For E2E-protected Method responses using profiles P04m, P07m, P08m, or P44m, in case of an error response (i.e., either an application error response message or an E2E error response message), STD_MESSAGERESULT_ERROR (1) shall be passed as argument messageResult to E2E_protect.⌋

[SWS_CM_90469] E2E Counter in E2E_protect for method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method responses, the E2E counter contained in the corresponding Method request shall be used as E2E counter in the call to E2E_protect.⌋

Note: The Method response carries the same dataID and E2E counter as the corresponding Method request to simplify the multiple client scenarios and allow the client to monitor the E2E counter.

[SWS_CM_90470] E2E protection header according to the network binding in the method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method responses, the E2E protection header shall be added to the message. If the protocol specification of the respective network binding imposes restrictions on the placement of the E2E protection header (e.g., [PRS_SOMEIP_00941] in case of SOME/IP network binding), then these restrictions shall be honored.⌋

7.4.1.2.5 E2E checking the service method response (Client)

[SWS_CM_90471] E2E checking of the method response in the ServiceProxy

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method responses, E2E checking shall be performed within the context of the message reception within the ServiceProxy.⌋

Figure 7.23 shows an overview of the interaction of components involved during the E2E checking of the Method response at the client side.

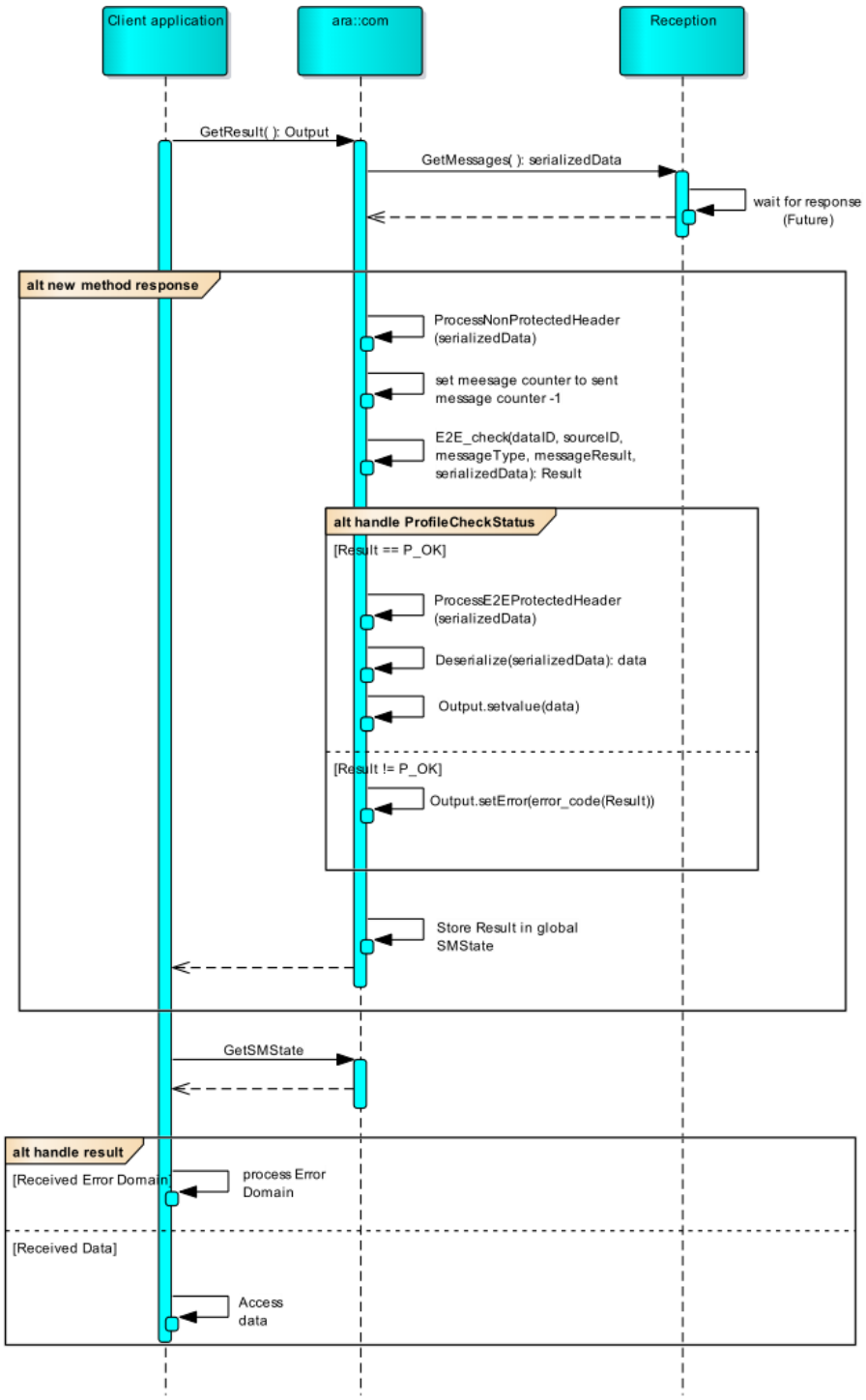

Figure 7.23: Interaction of components during E2E checking of the `Method` response at the client side

7.4.1.2.5.1 E2E checking of the payload

For E2E-protected Method responses, in case serialized data are available the following steps are to be done:

[SWS_CM_90472] Processing the non-E2E-protected header of the E2E-protected method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected Method responses, the non-E2E-protected header (if any) of the Method response's serialized data shall be processed.⌋

[SWS_CM_90473] Argument serialized Data in E2E_check for method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected Method response, E2E_check() shall be invoked on the protected serialized data (passed as argument serializedData to E2E_check()) according to [RS_E2E_08541], [PRS_E2E_00323], and [PRS_E2E_00828].⌋

[SWS_CM_90474] Argument dataId in E2E_check for method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected Method response, the End2EndMethodProtectionProps.dataId shall be passed as argument dataID to E2E_check()).⌋

[SWS_CM_10465] E2E counter of method response shall match with the one in method request

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method response, the response message shall carry the same E2E counter value as the request message. In case the E2E counter is different, the response message shall be discarded (without any further processing).⌋

Implementation Hint: The E2E counter can be extracted from the resulting state of the E2E_Protect()/E2E_Check() function.

[SWS_CM_90496] Argument sourceId in E2E_check for method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected Method responses using profiles P04m, P07m, P08m, or P44m, the End2EndMethodProtectionProps.sourceId shall be passed as argument sourceID to E2E_check.⌋

[SWS_CM_90497] Argument messageType in E2E_check for methods response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` responses using profiles P04m, P07m, P08m, or P44m, `STD_MESSAGETYPE_RESPONSE` (`1`) shall be passed as argument `messageType` to `E2E_check`.⌋

[SWS_CM_90498] Argument messageResult STD_MESSAGERESULT_OK in E2E_check for method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For E2E-protected `Method` responses using profiles P04m, P07m, P08m, or P44m, in case of a normal response (i.e., neither an application error response message nor an E2E error response message), `STD_MESSAGERESULT_OK` (`0`) shall be passed as argument `messageResult` to `E2E_check`.⌋

[SWS_CM_90499] Argument messageResult STD_MESSAGERESULT_ERROR in E2E_check for method response

Upstream requirements: RS_CM_00401, RS_E2E_08541

⌈For E2E-protected `Method` responses using profiles P04m, P07m, P08m, or P44m, in case of an error response (i.e., either an application error response message or an E2E error response message), `STD_MESSAGERESULT_ERROR` (`1`) shall be passed as argument `messageResult` to `E2E_check`.⌋

[SWS_CM_90478] E2E_check for method response provides Result with `ara::com::e2e::SMState` and `ara::com::e2e::ComE2EErrc`

Upstream requirements: RS_E2E_08541, RS_E2E_08534

⌈In return, for the given E2E-protected `Method` response, `E2E_check` shall provide a `Result` (`e2eResult` according to [PRS_E2E_00322] of [7]) containing the elements `ara::com::e2e::SMState` (`e2eState` according to [PRS_E2E_00322] of [7] mapped to the `ara::com::e2e::SMState` enum literal according to [SWS_CM_12022]) and `ara::com::e2e::ComE2EErrc` (`e2eStatus` according to [PRS_E2E_00322] of [7] mapped to the `ara::com::e2e::ComE2EErrc` enum literal according to [SWS_CM_12026]).⌋

[SWS_CM_90482] Update `ara::com::e2e::SMState` of specific method class with the `ara::com::e2e::SMState` provided in the Result of E2E_check

Upstream requirements: RS_CM_00400, RS_E2E_08541, RS_E2E_08534

⌈The global `ara::com::e2e::SMState` within its specific `Method` class of a specific `ServiceProxy` class shall be updated/overwritten with the element `ara::com::e2e::SMState` of the `Result` provided by `E2E_check` according to [SWS_CM_90478].⌋

[SWS_CM_90475] E2E protection header removal from the serialized data for method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected `Method` response, the E2E protection header shall be removed from the serialized data.⌋

7.4.1.2.5.2 Deserializing the payload

In case the call to `E2E_check` (according to [SWS_CM_90473]) indicated a successful E2E check of the response message, further processing of the response message shall take place.

[SWS_CM_90476] Deserialization of the data according to the network binding for method response

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected `Method` response, the resulting serialized data shall be de-serialized according to the rules of the respective network binding (e.g., according to [SWS_CM_10316] and [SWS_CM_10429] in case of SOME/IP network binding), resulting in the deserialized `inout` and `out` arguments to the `Method` call or in the de-serialized application error.⌋

[SWS_CM_10473] Handling the E2E Error Response

Upstream requirements: RS_CM_00223, RS_CM_00400, RS_E2E_08541

⌈Handling of an E2E error response message (sent due to a detected E2E error in request according to [SWS_CM_10472]) shall be done in the same way as the reception and the handling of any other error response message according to the used network binding (e.g., according to [SWS_CM_10429] in case of SOME/IP network binding).⌋

7.4.1.2.5.3 E2E error notification

In case the call to `E2E_check` (according to [SWS_CM_90473]) indicated a failed E2E check of the response message, the client application shall get notified in the following way:

[SWS_CM_90477] E2E Error Return Code

Upstream requirements: RS_CM_00400, RS_E2E_08541

⌈For the given E2E-protected `Method` response in case of failed E2E check an `ara::core::ErrorCode` of error domain `ara::com::e2e::ComE2EErrorDomain` with `value` set to `ara::com::e2e::ComE2EErrc` obtained in [SWS_CM_90478] shall

be constructed according to [SWS_CM_10474]. This ara::core::ErrorCode shall be passed as argument in a call to SetError() on the ara::core::Promise.⌋

The handling of normal and application error responses (according to [SWS_CM_90476]) combined with the handling of E2E error responses (according to [SWS_CM_10473]) and the explicit notification of E2E errors detected in the response message (according to [SWS_CM_90477]) will yield an ara::core::Result containing either

- the correct output of the server operation in case of absence of any error
- an ara::core::ErrorCode of the error domain ApApplicationError.errorDomain with the value set to ApApplicationError.errorCode of the raised ApApplicationError in case the ClientServerOperation raised one of its configured possible ClientServerOperation.possibleApErrors and no E2E error was detected in the request message and the response message
- an ara::core::ErrorCode of error domain ara::com::e2e::ComE2EErrorDomain and the value set to the ara::com::e2e::ComE2EErrc of the Result of the E2E_check call at the server side in case an E2E error was detected in the request message at the server side and no E2E error was detected in the response message at the client side
- an ara::core::ErrorCode of error domain ara::com::e2e::ComE2EErrorDomain and the value set to the ara::com::e2e::ComE2EErrc of the Result of the E2E_check call at the client side in case an E2E error was detected in the response message at the client side

[SWS_CM_90483] GetE2EStateMachineState method shall be provided for each method class

Upstream requirements: RS_E2E_08534

⌈A GetE2EStateMachineState method shall be provided for each Method class of a specific ServiceProxy class.⌋

[SWS_CM_90484] GetE2EStateMachineState method shall provide access to the ara::com::e2e::SMState of the specific method class

Upstream requirements: RS_E2E_08534

⌈The GetE2EStateMachineState method shall provide access to the global ara::com::e2e::SMState of the specific Method class, which was determined by the last run of E2E_check function invoked during the last reception of the Method response (see [SWS_CM_90482]).⌋

```
1 ara::com::e2e::SMState GetE2EStateMachineState() const noexcept;
```

7.4.1.2.6 Timeout supervision

ara::com does not support any timeout supervision for method calls. A lost response message could block some ara::core::Future methods like wait() forever. In case of E2E such a timeout supervision is desired, wherefore the adaptive application is strongly recommended to implement timeout supervision, e.g., by using the ReportCheckpoint() method of the ara::phm::SupervisedEntity or the wait_for(), wait_until(), or the is_ready() methods of the ara::core::Future.

7.4.1.3 Fields

This section specifies E2E protection for fields. For details of fields see [4]. A field is a data object that can be accessed by a getter and/or setter method. In addition update notifications may be provided to subscribers, whenever the value of the field gets updated. The principle of fields is already specified. This section specifies the E2E protection for fields. The E2E protection for methods Get and Set follows the E2E protection for Methods (chapter 7.4.1.2). The specifications [SWS_CM_10460] and [SWS_CM_90485] define the parameters for E2E protection of the methods Get() and Set(). The limitations of chapter 7.4.1.2.1 are applicable.

The E2E protection for Update follows the E2E protection for events (chapter 7.4.1.1). The specifications [SWS_CM_90402] and [SWS_CM_90433] define the parameters for E2E protection of the update event. The limitations of chapter 7.4.1.1.1 are applicable.

E2E results OK and OK_SOME_LOST are successful results. E2E results ERROR, REPEATED, WRONGSEQUENCE and NONEWDATA are considered error results.

There are E2E profiles 4m, 7m, 8m or 44m for the protection of methods (Get(), Set()). Also the other E2E profiles can be used for the protection of Methods. But in this case some parameters of SOME/IP are not protected.

7.4.1.3.1 Send a GET message

The client application calls the Get() function at ara::com without arguments. A ara::core::Future for this method call is created by ara::com. Data of method Get() are serialized.

The E2E serialization follows the specification of [SWS_CM_00041] with the following exception: The result is a list without parameters because a Get() method has no IN or INOUT parameters.

The parameters dataID, sourceID, messageType and messageResult for E2E_XXmProtect method are passed as described in chapter 7.4.1.2.2.2.

After E2E protection the non E2E protected part is added to the message as described in [SWS_CM_10464].

Figure 7.24 shows the message flow of sending a `Get()` method. The figure does not list all details of E2E protection, e.g. functions of CRC library are omitted in this figure.

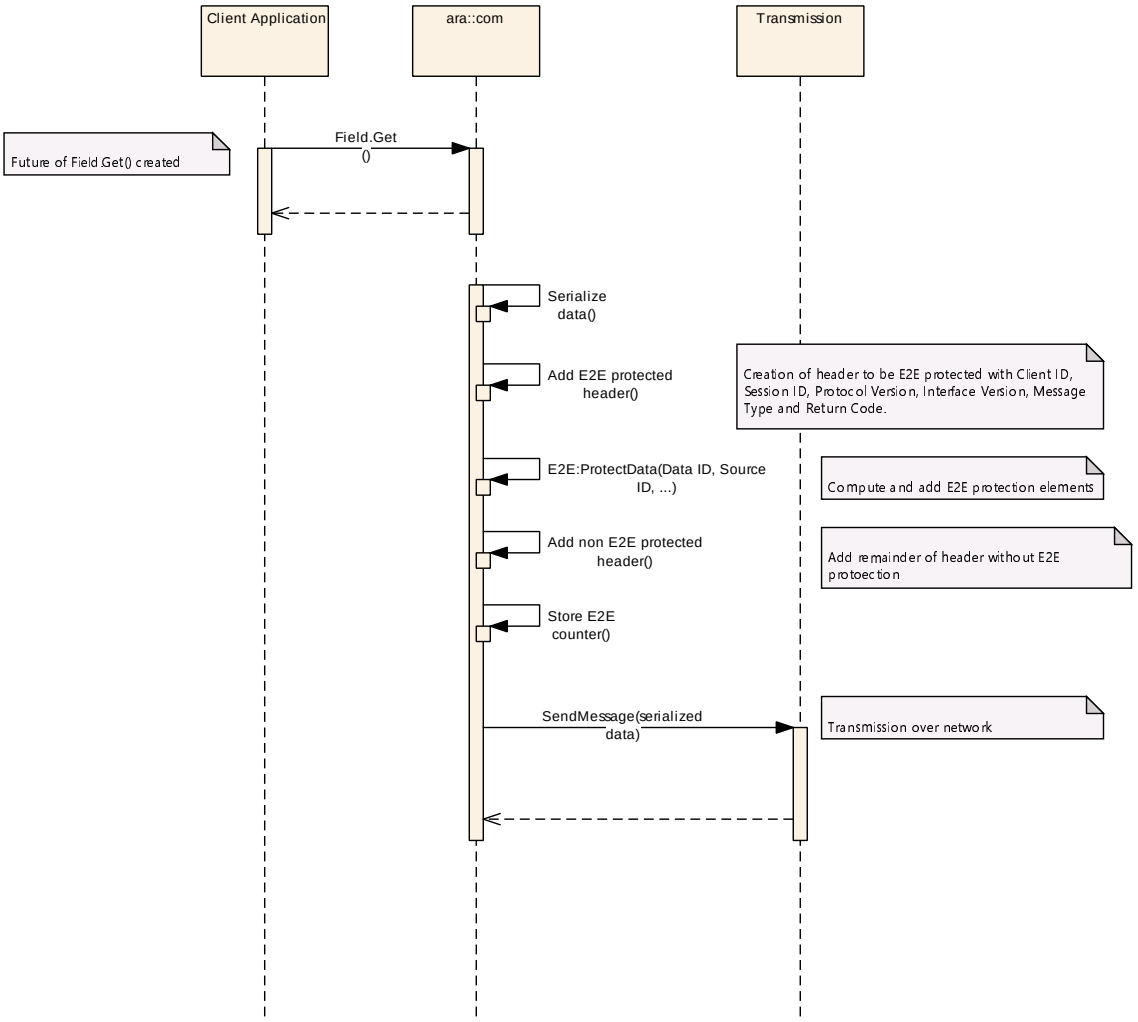

Figure 7.24: Send a GET Message

7.4.1.3.2 Receive a GET message

The message is received by the Publisher application. The Publisher application is a server application.

The E2E check of the received message follows the specification of chapter 7.4.1.2.3.

The type of the message to be sent back to the client is `RESPONSE` or `ERROR`. That depends on the result of the E2E check. If the E2E check fails, then the `Return Code` of the `ERROR` message is initialized with an E2E error code (See [PRS_SOMEIP_00191]).

Figure 7.25 shows the reception of a GET message. The E2E protected part of the serialized header is checked for E2E errors. If the incoming message was received with an E2E error, then the Publisher is informed through the E2E error handler (see chapter 7.4.1.2.3.3). In this case no value is retrieved from Publisher.

If the incoming message is received without E2E error, the GetHandler of the Publisher application is called.

Independent of the result of the E2E check a response message is sent to the client (caller of the `Get()`() function). The message sent back to the client has message type type `RESPONSE` and return code either (`OK`) or (`ERROR`).

This response message is E2E protected the same way as the `Get()` message as described in chapters 7.4.1.2.4.1, 7.4.1.2.4.2 and 7.4.1.2.4.3.

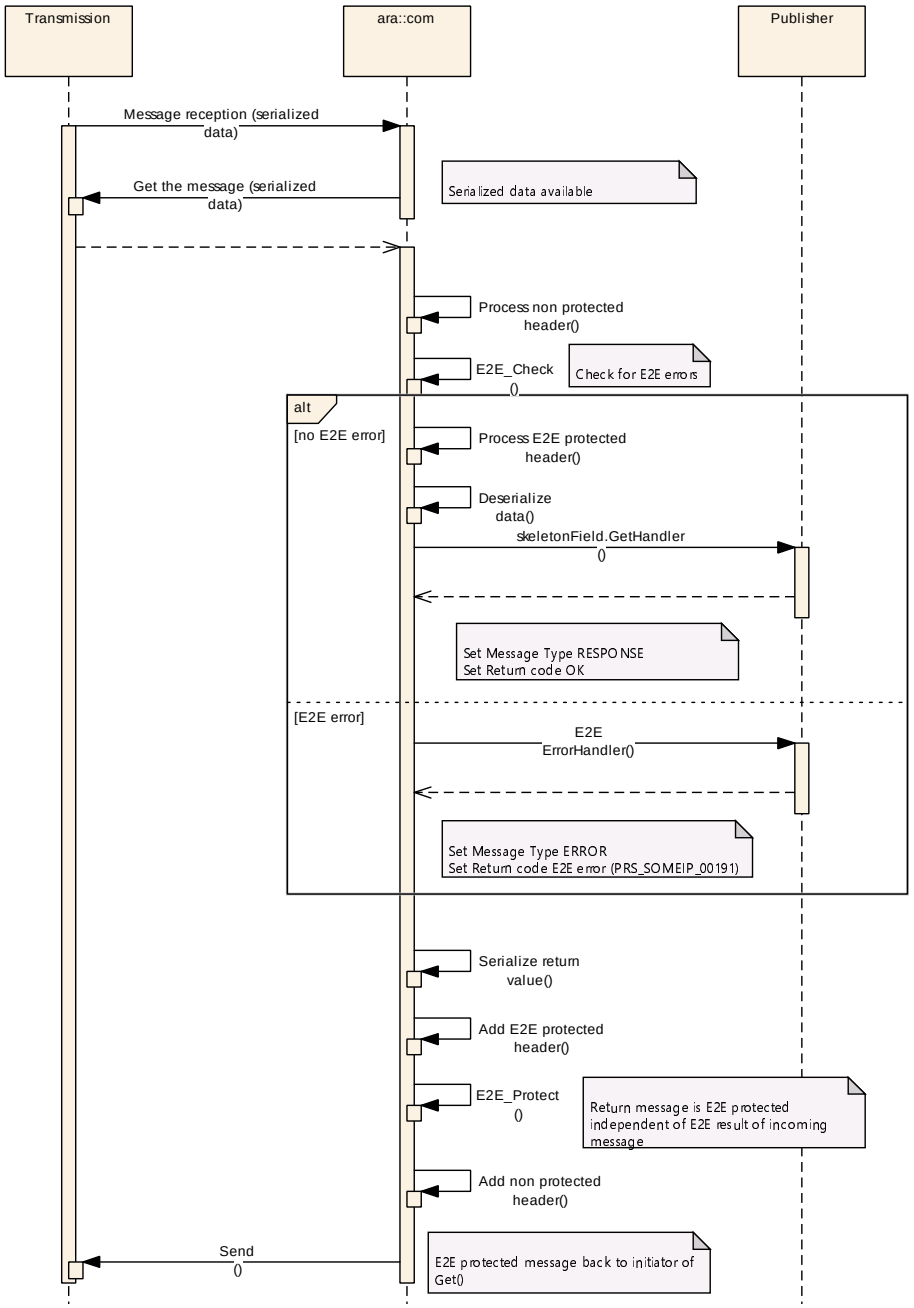

Figure 7.25: Receive a GET Message

7.4.1.3.3 Receive a response to a GET message

The reception of an E2E protected response message is described in chapter 7.4.1.2.5.

If the message is received with an E2E error, then the E2E Errorhandler of the client is called. The `ara::core::Future` of the `Get()` function is set to ready state with an error code. That is described in chapter 7.4.1.2.5.

The received message is of type RESPONSE or ERROR (see [PRS_SOMEIP_00055]). Type ERROR indicates that an E2E error occurred at the server site. If a message of type ERROR is received with Return Code of E2E error (indicating that the Publisher received the Get() request with an E2E error) then the E2E Errorhandler of the Client Application is called. The ara::core::Future of the Get() function is set to ready state with an error code.

It is up to the Client application how to react to a call of its Errorhandler.

If the RESPONSE message is received without E2E errors then the ara::core::Future is updated with the received value of the Publishers field. The ara::core::Future becomes ready and the Client application can use this value.

If a RESPONSE message to an outgoing Get() message does not arrive at all, then the client application is not informed if the value was retrieved from the remote application. The ara::core::Future of Field.Get() is not updated to state ready. In this case the client application can send the Get() message again to the remote application to retrieve the value, or initiate its own error handling. A timeout supervision (chapter 7.4.1.2.6) may unlock the ara::core::Future. Figure 7.26 shows reception of a message from the server.

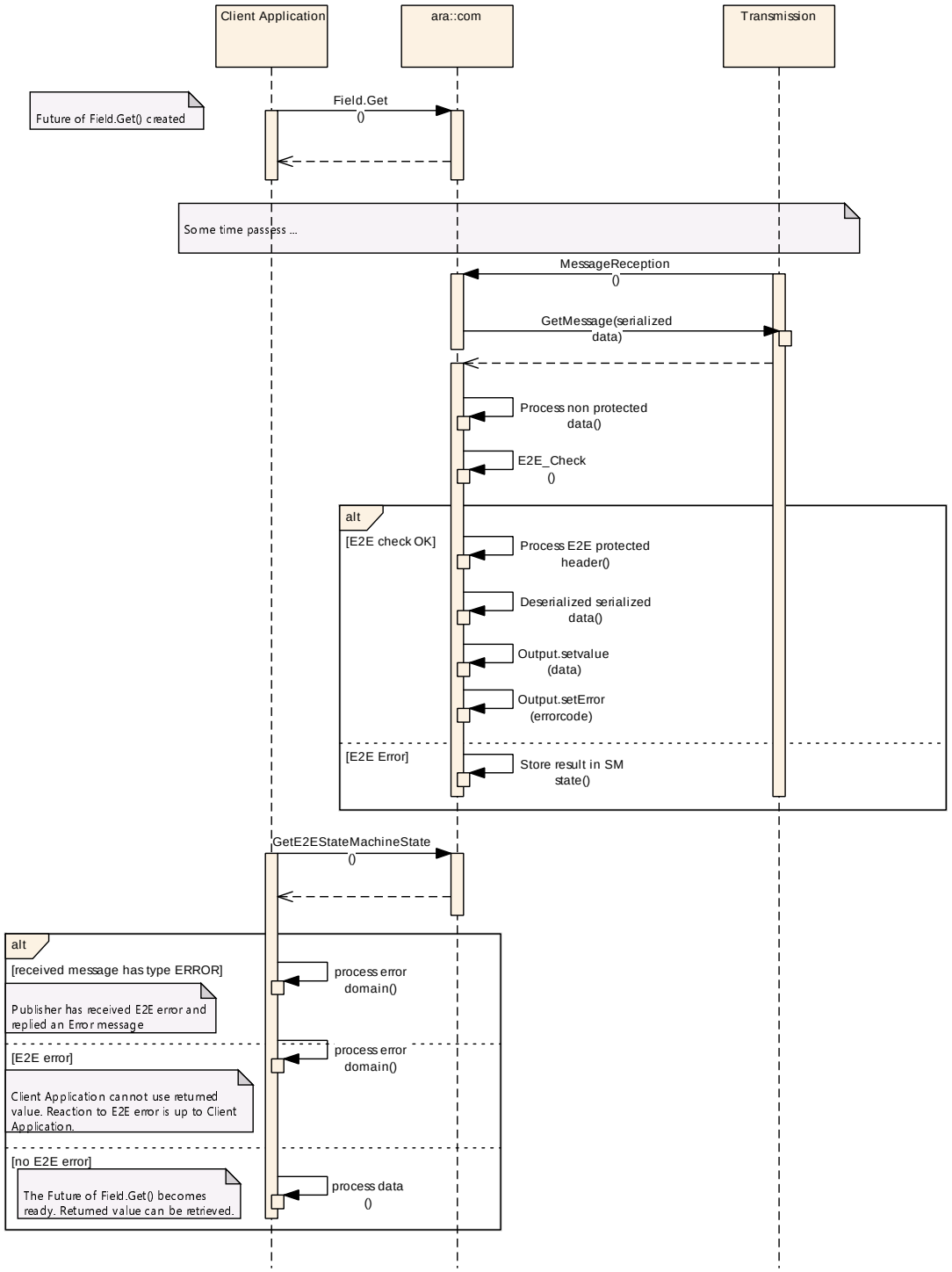

Figure 7.26: Receive response to a GET Message

7.4.1.3.4 Send a SET message

The E2E serialization follows the specification of [SWS_CM_00041]. Only one parameter is serialized: The parameter to be set at the publisher application.

The parameters `dataID`, `sourceID`, `messageType` and `messageResult` for `E2E_XXmProtect` method are passed as described in chapter 7.4.1.2.2.2.

After E2E protection the non E2E protected part is added to message as described in specification [SWS_CM_10464].

Figure 7.27 shows the message flow of sending a `Set()` method. The figure does not list all details of E2E protection, e.g. functions of libraries E2ELib and CrcLib are omitted in this figure.

The client application calls the `Set()` function at `ara::com` with one argument (the value that shall overwrite the field's value).

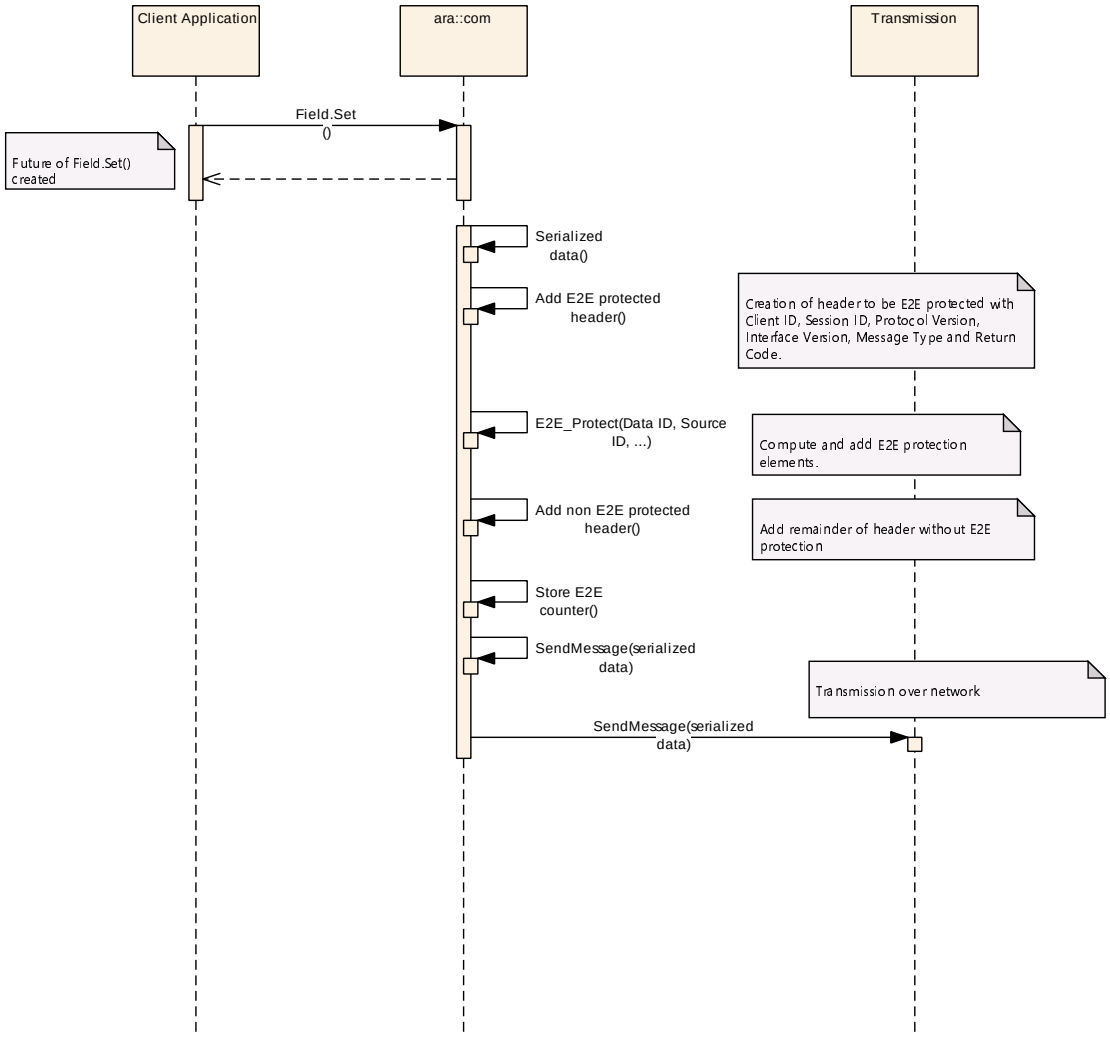

Figure 7.27: Send a SET Message

7.4.1.3.5 Receive a SET message

The message is received by the Publisher application. The Publisher application is a server application.

The E2E check of the received message follows the specification of chapter 7.4.1.2.3.

If the incoming message is received without E2E error the SetHandler of the Publisher application is called. The SetHandler returns the value to be written to the Publisher's field. The returned value may be identical to the parameter of the `Set()` message (successful update). But there is also the possibility that an update could not be performed completely. If the parameter of the `Set()` message is out of range then the field may be left unchanged or the field is updated by a value inside the field's range. The type of the response message is `RESPONSE`.

If the incoming message is received with an E2E error, then the Publisher is informed through the E2E error handler (see chapter 7.4.1.2.3.3). In this case The SetHandler of the Publisher is not called. The type of the response message is `ERROR`. If the `E2E_Check` fails the `Return Code` of the `ERROR` message is initialized with an E2E error code (See [PRS_SOMEIP_00191]).

The type of the message to be sent back to the client is `RESPONSE` or `ERROR`. That depends on the result of the E2E check.

The message to be returned (type `ERROR` or `RESPONSE`) is serialized, E2E protected and sent back to the client.

This response message is E2E protected the same way as the `Get()` message as described in chapters 7.4.1.2.4.1, 7.4.1.2.4.2 and 7.4.1.2.4.3.

Figure 7.28 shows the reception of a `Set()` message. The E2E protected part of the serialized header is checked for E2E errors. If the incoming message was received with an E2E error, then the Publisher is informed through the E2E error handler. The Publisher's field is not updated and no value is retrieved from Publisher's field.

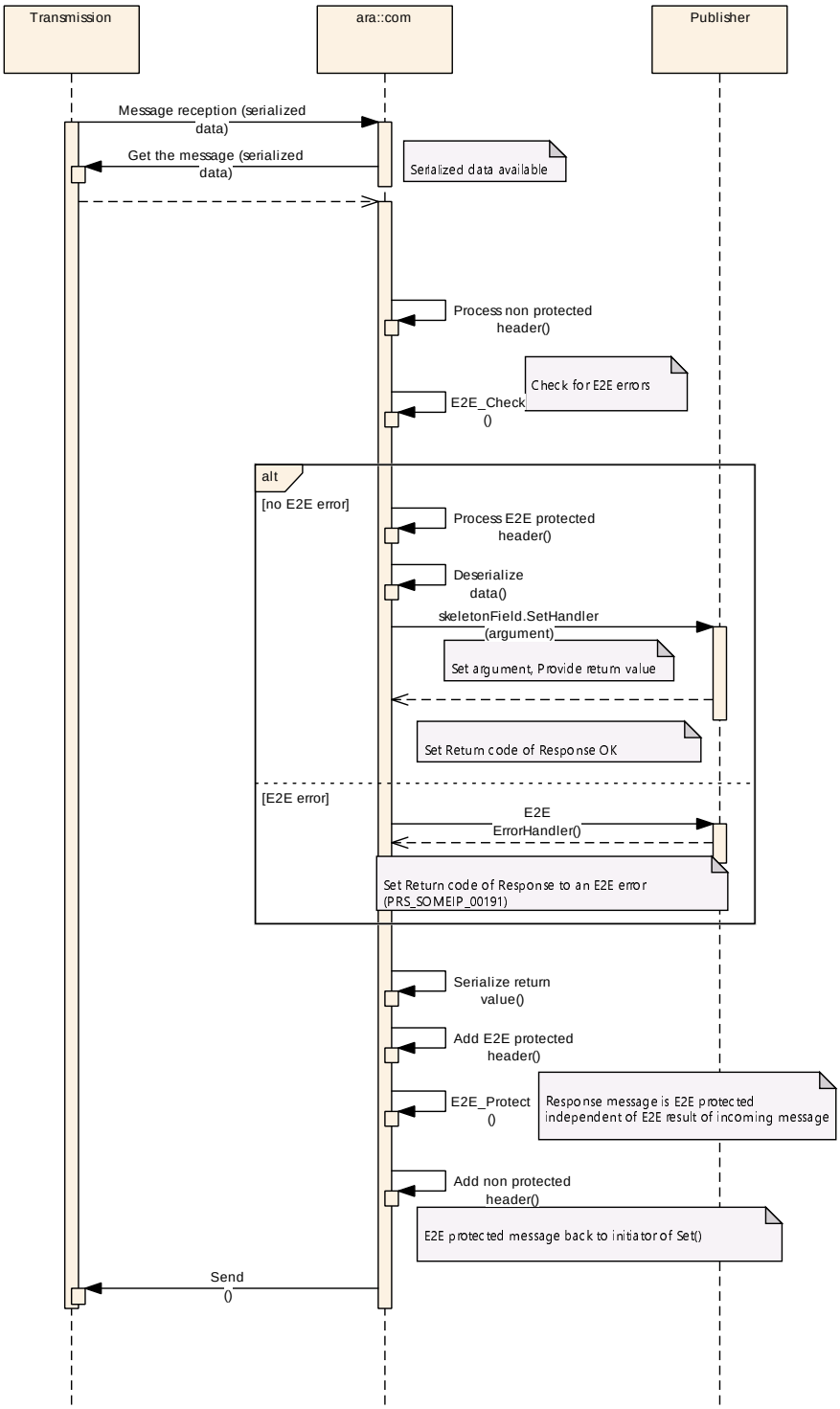

Figure 7.28: Receive a SET Message

7.4.1.3.6 Receive a response to a SET message

The reception of an E2E protected response message is described in chapter 7.4.1.2.5.

If the message is received with an E2E error, then the Errorhandler of the client is called. The future of the `Set()` function is set to ready state with an error code ().That is described in chapter 7.4.1.2.5.3.

The received message is of type `RESPONSE` or `ERROR` (see [PRS_SOMEIP_00055]). Type `ERROR` indicates that an E2E error occurred at the server site. If a message of type `ERROR` is received with `Return Code` of E2E error (indicating that the Publisher received the `Set()` request with an E2E error) then the Errorhandler of the Client Application is called. The `ara::core::Future` of the `Set()` function is set to ready state with an error code.

It is up to the Client application how to react to a call of its Errorhandler.

If the `RESPONSE` message is received without E2E errors then the `ara::core::Future` is updated with the received value of Publisher's field. The `ara::core::Future` becomes ready and the Client application can use this value.

If a `RESPONSE` message to an outgoing `Set()` message does not arrive at all then the client application is not informed about the value which is set at the remote application. The `ara::core::Future` of `Set()` is not updated to state ready. In this case the client application can send the `Set()` message again to the remote application in order to set the intended value and receive the set value or initiate its own error handling. A timeout supervision (chapter 7.4.1.2.6) can unlock the `ara::core::Future`.

Figure 7.29 shows reception of a response. This message is of type `RESPONSE` or `ERROR` (see [PRS_SOMEIP_00055]) and similar to the reception of a response to a `Get()` message.

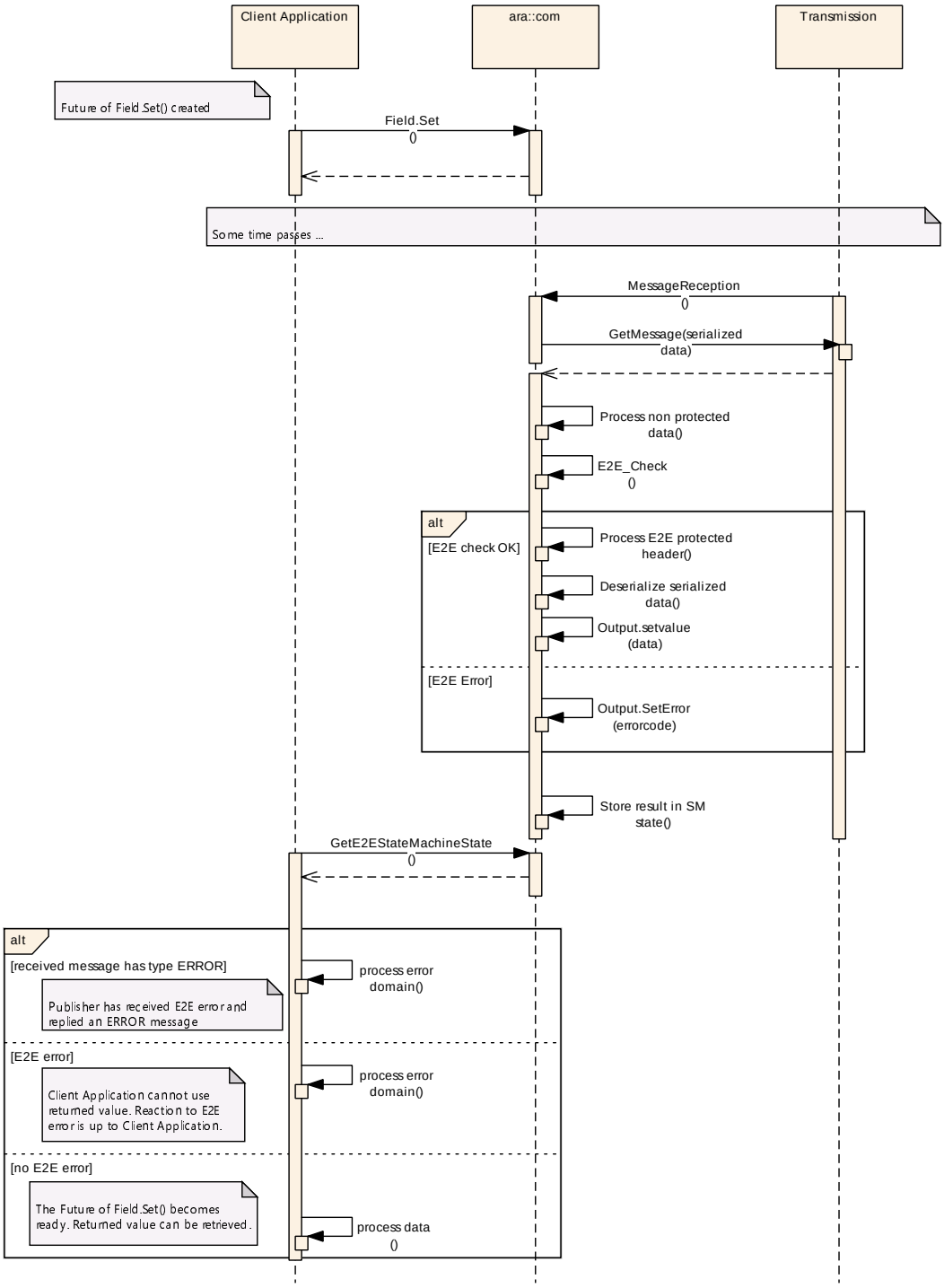

Figure 7.29: Receive response to a SET Message

7.4.1.3.7 Send an UPDATE message

The application triggers the sending of update messages to subscribers. The update of a field's value by a SetHandler() is a reason to trigger update messages.

An update of a subscriber is an event. The E2E protection of an update is described in chapter 7.4.1.1.2. The update message is sent to every subscriber to the publisher's field.

Figure 7.30 shows sending of field update messages.

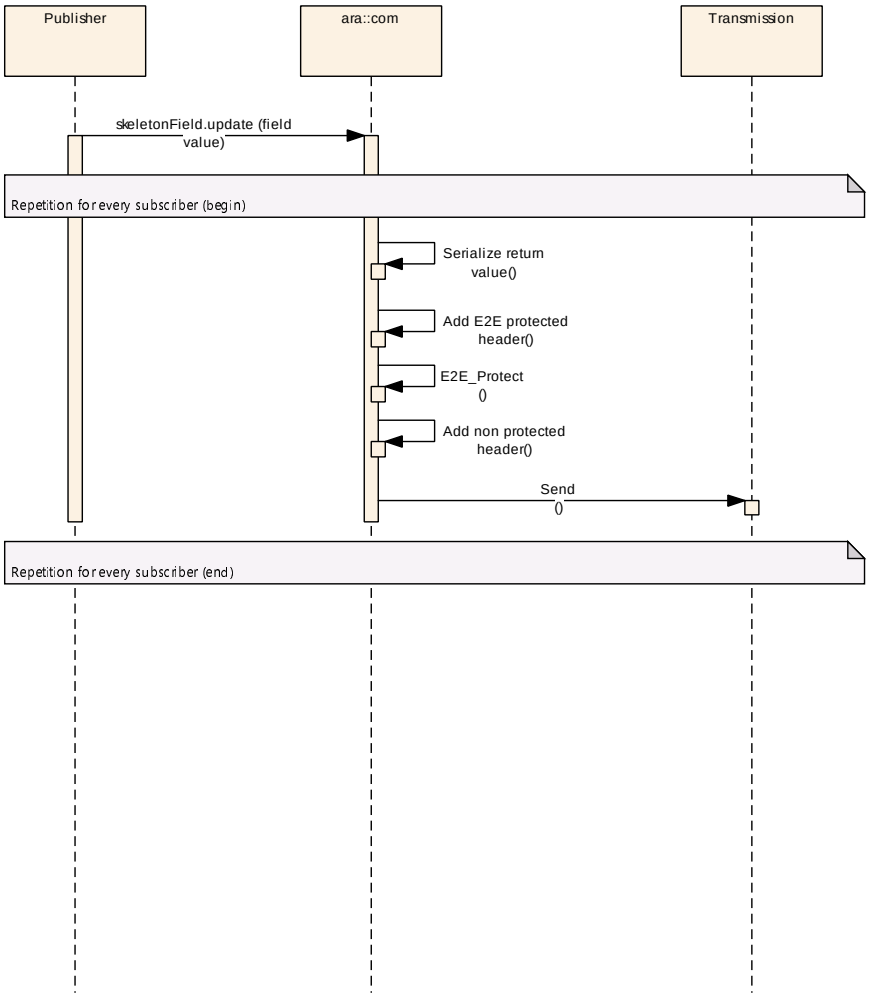

Figure 7.30: Send an UPDATE Message

7.4.1.3.8 Receive an UPDATE message

The loop over samples indicates that more than one update messages are collected and evaluated by E2E state machine. In the case of E2E fields this is rather a theoretical option. Usually the number of received update messages is zero or one.

The reception of E2E protected fields is described in chapter 7.4.1.1.3.

The reception of E2E protected fields follows the principle of E2E protected events (see figure 7.18 in chapter 7.4.1.1). This reception of E2E protected fields demands periodic communication.

If one or more update messages are received the E2E state machine provides one of the following results: `OK`, `ERROR`, `REPEATED`, `NONEWDATA`, `WRONGSEQUENCE` (See [PRS_E2E_00597]). Only result `OK` indicates that the received value is valid.

Figure 7.31 shows reception of a field update message.

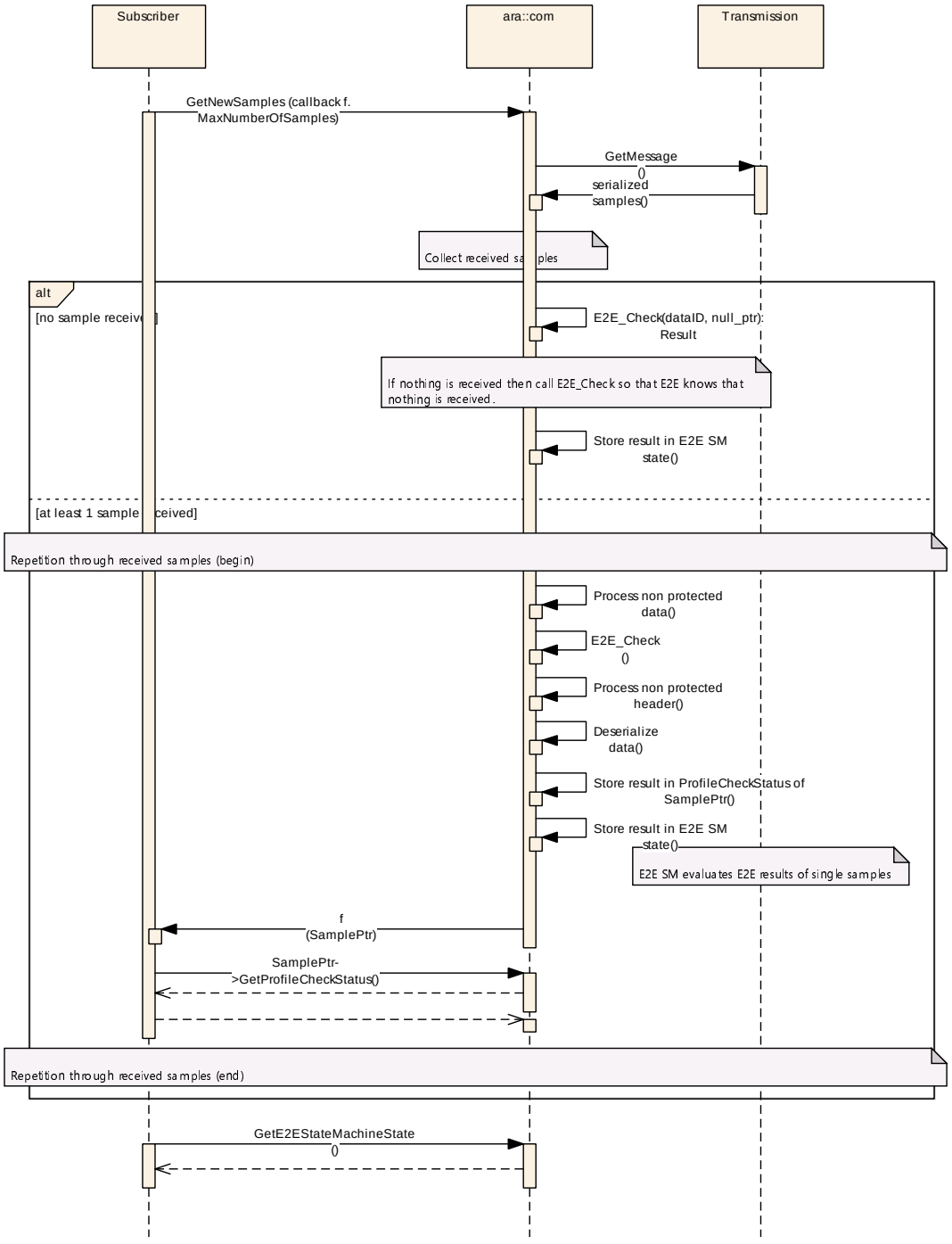

Figure 7.31: Receive an UPDATE Message

7.4.2 End-to-end communication protection for DDS

The present DDS network binding is defined in terms of interactions between `ara::com` APIs and standard DDS APIs. Hence, End-to-end communication protection as described in sections 7.4.1.1 and 7.4.1.2 doesn't apply, because API calls can't be checksummed or payloaded the same way serialized messages are.

By no means does this imply that DDS is exempt from E2E protection assurances, they are simply provided by the DDS middleware. Please find below the different kinds of faults defined in [7] (derived from ISO-26262-6:2011, annex D.2.4) and their corresponding DDS/RTPS protection mechanism:

- Repetition, loss, insertion, incorrect sequence, information from a sender received by only a subset of receivers, and blocking access to a communication channel: as described in [FO_PRS_DDS_00601] and [FO_PRS_DDS_00602]
- Delay of information and blocking access to a communication channel: as described in [FO_PRS_DDS_00603]
- Masquerade or incorrect addressing of information: DDS Security authentication plugin, as defined in [26] section 8.3 "Authentication Plugin"
- Corruption of information, asymmetric information sent from a sender to multiple receivers: as described in [FO_PRS_DDS_00604]
- Translation of these fault conditions into `ara::com::e2e::ProfileCheckStatus` values depends on the specific capacities of the DDS implementation to report per-sample the status of the aforementioned protection measures (sequence numbers, latency budget, message authentications, checksums)

The following sections specifiy the integration of E2E communication protection in the DDS Network Binding.

Configuration of End-to-end protection mechanisms is achieved via DDS [20] QoS policy configuration and the DDS Wire Interoperability protocol [21], in opposition to Manifest [5] configuration and the AUTOSAR E2E protocol [7] as described by [SWS_CM_90402], [SWS_CM_90433] and [SWS_CM_90417].

This Transport Layer (or "Layer 4" according to the OSI model) approach brings additional safety features (like sample rejection, delay or loss) which `TransportFaultCondition`-related `ara::com` APIs aim to enable for Adaptive Applications. Support of these APIs from the different `ara::com` Network Bindings is optional.

7.4.2.1 Events

This section specifies the integration of E2E communication protection in the DDS Network Binding of `ara::com` for the processing of `Event`s.

[SWS_CM_00210] Mapping of Event::GetE2EStateMachineState() for Proxy Events

Status: DRAFT

⌈`Event::GetE2EStateMachineState()` shall return one of the following:

- `SMState::kStateMDisabled` if the underlying `DataReader` `LIVELINESS` QoS policy has `lease_duration` set to `DDS_DURATION_INFINITE`
- `SMState::kInvalid` if the underlying `DataReader` has not yet signaled `DDS_LIVELINESS_CHANGED_STATUS`, or: it has signaled `DDS_LIVELINESS_CHANGED_STATUS` and `alive_count` has decreased to zero
- `SMState::kValid` if the underlying `DataReader` has signaled `DDS_LIVELINESS_CHANGED_STATUS` with `alive_count` higher than zero
- `SMState::kIncompatibleQoS` if the underlying `DataReader` has signaled `DDS_REQUESTED_INCOMPATIBLE_QOS_STATUS`

The remaining `ara::com::e2e::SMState` values (`kInit` and `kNoData`) are not supported by `Event::GetE2EStateMachineState()` within the DDS Network Binding.

⌋

[SWS_CM_00211] Mapping of ara::com::e2e::TransportFaultCondition for Proxy Events

Status: DRAFT

⌈`ara::com::e2e::TransportFaultConditionHandler` configured via `Event::SetTransportFaultConditionHandler()` shall report one of the following values for each transport error detected:

- `TransportFaultCondition::kSampleRejected` if the underlying `DataReader` entity has signaled `SAMPLE_REJECTED` (e.g. due to corruption or lack of storage resources)
- `TransportFaultCondition::kDeadlineMissed` if the underlying `DataReader` has signaled `REQUESTED_DEADLINE_MISSED` (e.g. due to one or more missed timing deadlines)
- `TransportFaultCondition::kSampleLost` if the underlying `DataReader` has signaled `SAMPLE_LOST` (e.g. because one or more samples have been detected to be lost)

⌋

7.4.2.2 Triggers

[SWS_CM_00212] Mapping of ara::com::e2e::TransportFaultCondition for Proxy Triggers

Status: DRAFT

⌈`ara::com::e2e::TransportFaultConditionHandler` configured via `Trigger::SetTransportFaultConditionHandler()` shall report one of the following values for each transport error detected:

- `TransportFaultCondition::kSampleRejected` if the underlying `DataReader` entity has signaled `SAMPLE_REJECTED` (e.g. due to corruption or lack of storage resources)
- `TransportFaultCondition::kDeadlineMissed` if the underlying `DataReader` has signaled `REQUESTED_DEADLINE_MISSED` (e.g. due to one or more missed timing deadlines)
- `TransportFaultCondition::kSampleLost` if the underlying `DataReader` has signaled `SAMPLE_LOST` (e.g. because one or more samples have been detected to be lost)

⌋

7.4.2.3 Methods

This section specifies the integration of E2E communication protection in the DDS Network Binding of `ara::com` for the processing of `Method`s.

[SWS_CM_00213] Skeleton E2E Error Handler - Invocation

Status: DRAFT

⌈`Skeleton::E2EErrorHandler()` shall be invoked from within a separate thread by the Communication Management software in case the underlying `DataReader` entity for method requests reports an E2E error.⌋

[SWS_CM_00214] Skeleton E2E Error Handler - Invocation Arguments

Status: DRAFT

⌈In case the underlying `DataReader` entity for method requests reports an E2E error, E2EErrorHandler shall be called with the following arguments:

- `errorCode` set to the `ProfileCheckStatus` derived from the `DataReader` entity status, similar to [SWS_CM_00210]
- `dataId` set the method's unique ID `<svcId>Method_<methodName>_Hash`, as defined by section 7.5.1.1.6 of [21]

- messageCounter shall be set to the publication sequence number if available, zero otherwise

⌋

[SWS_CM_00215] Mapping of Method::GetE2EStateMachineState() for Proxy Methods

Status: DRAFT

⌈Method::GetE2EStateMachineState() shall return one of the following:

- SMState::kStateMDisabled if the underlying DataReader LIVELINESS QoS policy has lease_duration set to DDS_DURATION_INFINITE
- SMState::kInvalid if the associated DataReader entity has not yet signaled LIVELINESS_CHANGED_STATUS, or: it has signaled LIVELINESS_CHANGED_STATUS and alive_count has decreased to zero
- SMState::kValid if the associated DataReader entity has signaled LIVELINESS_CHANGED_STATUS with alive_count higher than zero
- SMState::kIncompatibleQoS if the underlying DataReader has signaled REQUESTED_INCOMPATIBLE_QOS_STATUS

The remaining ara::com::e2e::SMState values (kInit and kNoData) are not supported by Event::GetE2EStateMachineState() within the DDS Network Binding.

⌋

[SWS_CM_00216] Mapping of ara::com::e2e::TransportFaultCondition for Proxy Methods

Status: DRAFT

⌈ara::com::e2e::TransportFaultConditionHandler configured via Skeleton::SetTransportFaultConditionHandler() shall report one of the following values for each transport error detected:

- TransportFaultCondition::kSampleRejected if the underlying request DataReader entity has signaled SAMPLE_REJECTED (e.g. due to corruption or lack of storage resources)
- TransportFaultCondition::kDeadlineMissed if the underlying request DataReader has signaled REQUESTED_DEADLINE_MISSED (e.g. due to one or more missed timing deadlines)
- TransportFaultCondition::kSampleLost if the underlying request DataReader has signaled SAMPLE_LOST (e.g. because one or more samples have been detected to be lost)

⌋

7.4.2.4 Fields

This section specifies the integration of E2E communication protection in the DDS Network Binding of `ara::com` for the processing of `Field`s.

[SWS_CM_00217] Mapping of Field::GetE2EStateMachineState() for Proxy Field Notifiers

Status: DRAFT

⌈Field Notifiers shall implement `GetE2EStateMachineState()` according to [SWS_CM_00210].⌋

[SWS_CM_00218] Mapping of ara::com::e2e::TransportFaultCondition for Proxy Field Notifiers

Status: DRAFT

⌈Field Notifiers shall implement `ara::com::e2e::TransportFaultCondition` notifications according to [SWS_CM_00211].⌋

[SWS_CM_00219] Skeleton E2E Error Handler - Invocation and Invocation Parameters

Status: DRAFT

⌈Field Getter and Setter shall implement `Skeleton::E2EErrorHandler()` according to [SWS_CM_00213] and [SWS_CM_00214].⌋

[SWS_CM_00220] Mapping of Field::GetE2EStateMachineState() for Proxy Field Getter and Setter

Status: DRAFT

⌈Field Getter and Setter shall implement `Field::Get::GetE2EStateMachineState()` and `Field::Set::GetE2EStateMachineState()` according to [SWS_CM_00215].⌋

[SWS_CM_00221] Mapping of ara::com::e2e::TransportFaultCondition for Proxy Field Getter and Setter

Status: DRAFT

⌈Field Getters and Setters shall implement `ara::com::e2e::TransportFaultCondition` notifications according to [SWS_CM_00216].⌋

7.5 Communication Interfaces

ara::com is the interface that AUTOSAR Adaptive Applications use to interact with the Communication Management.

In this chapter, the functional specifications for the communication interfaces of ara::com are described. The actual C++ APIs of ara::com are described in chapter 8.

7.5.1 Offer service

For the service offering C++ API reference, see chapter 8.3.5.2.5.1 and 8.3.5.2.5.2.

[SWS_CM_00102] Uniqueness of offered service on local machine

Upstream requirements: RS_CM_00200, RS_CM_00101, RS_CM_00108

⌈Upon a call to `OfferService()` the Communication Management shall check the offered service for uniqueness on the local machine using information available to the service discovery. If the implementation detects a service instance duplication (i.e., a service with the same `serviceInstanceId`, `serviceInterfaceId` and `majorVersion` on the same VLAN (e.g.according to [constr_1723] of [5]) is already registered, the requested service offering shall not start, and the function shall return positively after error is logged.⌋

Note: System/vehicle-wide Uniqueness of offered service (see [RS_CM_00108]); System/vehicle-wide uniqueness should be targeted in a best-effort way, i.e., if knowledge about a a remotely offered service is available, this knowledge shall be used in the uniqueness check.

[SWS_CM_00103] Network binding where a service is offered

Upstream requirements: RS_CM_00101

⌈When a new service is offered by the application, the Communication Management shall check over which network binding this service shall be offered. This information is configured in the class of `ServiceInterfaceDeployment` referencing the offered `ServiceInterface` in the role `serviceInterface`. If the class is `SomeipServiceInterfaceDeployment` then the Some/IP network binding shall handle the `OfferService()` call as described in [SWS_CM_00203]. If the class is `DdsServiceInterfaceDeployment`, then the DDS network binding shall handle the `OfferService()` call as described in [SWS_CM_11001]. If the class is `UserDefinedServiceInterfaceDeployment`, the Communication Management implementer is responsible for implementing the `OfferService()` method in an appropriate way.⌋

[SWS_CM_00104] Network binding for StopOfferService

Upstream requirements: RS_CM_00101

⌈When a service calls `StopOfferService()`, the Communication Management shall check over which network binding the offered service shall be stopped. This information is configured in the class of `ServiceInterfaceDeployment` referencing the offered `ServiceInterface` in the role `serviceInterface`. If the class is `SomeipServiceInterfaceDeployment` then the Some/IP network binding shall handle the mapping of the `StopOfferService()` method as described in [SWS_CM_00204]. If the class is `DdsServiceInterfaceDeployment`, then the DDS network binding shall handle the mapping of the `StopOfferService()` as described in [SWS_CM_11005]. If the class is `UserDefinedServiceInterfaceDeployment`, the Communication Management implementer is responsible for implementing the `StopOfferService()` method in an appropriate way.⌋

7.5.2 Service skeleton creation

For the service skeleton creation C++ API reference, see chapter 8.3.5.2.2.3, 8.3.5.2.2.2, 8.3.5.2.2.1, 8.3.5.2.1.4, 8.3.5.2.1.2, 8.3.5.2.1.3, 8.3.5.2.1.1 and 8.3.5.2.1.5.

[SWS_CM_10410] `ara::com::InstanceIdentifier` check during the creation of service skeleton

Upstream requirements: RS_CM_00101

⌈The Communication Management shall check the value of the `ara::com::InstanceIdentifier` argument: the identifier shall be unique. If the same `ara::com::InstanceIdentifier` is used for the creation of more than one skeleton instance of the same service shall be handled as violation according to [SWS_CORE_00003].⌋

[SWS_CM_10450] `ara::core::InstanceSpecifier` check during the creation of service skeleton

Upstream requirements: RS_CM_00101, RS_AP_00137

⌈The Communication Management shall check the value of the `ara::core::InstanceSpecifier` argument: the specifier shall be unique, using the same instance specifier for the creation of more than one skeleton instance of the same service shall be handled as violation according to [SWS_CORE_00003].⌋

[SWS_CM_10451] `ara::com::InstanceIdentifierContainer` check during the creation of service skeleton

Upstream requirements: RS_CM_00101

⌈The Communication Management shall check the value of the `ara::com::InstanceIdentifierContainer` argument:

- the container size shall be bigger than zero
- the identifiers of the container shall be unique
- the identifiers of the container shall correspond to the same instance specifier.

If there are failing checks, and the same `ara::com::InstanceIdentifier` is used for the creation of more than one skeleton instance of the same service shall be handled as violation according to [SWS_CORE_00003].⌋

[SWS_CM_10467] Wrong Method Call Processing Mode Error for ServiceSkeleton constructor

Upstream requirements: RS_CM_00402, RS_CM_00400, RS_E2E_08541

⌈In case a `ara::com::MethodCallProcessingMode==kEvent` has been passed to the constructor of the `ServiceSkeleton` for a service using E2E-protected methods, it shall be handled as a `WrongMethodCallProcessingModeViolation`.⌋

Note: A `ara::com::MethodCallProcessingMode==kEvent` is not supported for E2E-protected `Method`s.

7.5.3 Service skeleton Destruction

For the service skeleton destruction C++ API reference, see [SWS_CM_11370].

[SWS_CM_00083] Call StopOfferService() before destruction of Skeleton ⌈If the skeleton object is destroyed while a method call is still in progress (using asynchronous handling, `MethodCallProcessingMode==kEvent`), the behavior is undefined. Therefore, `StopOfferService()` should be called either from the application before the destruction, or within the user defined destructor in the inherited skeleton class.⌋

Note: The implicit call of `StopOfferService()` in the destructor is only performed to make sure the service is not offered after the skeleton is destroyed. [SWS_CM_00083] still applies to avoid destruction failures.

[SWS_CM_00084] Readying the Future of a Method implementation after Skeleton destruction ⌈If the skeleton object is destroyed while a method call is still in progress, calls to `set_value()` (see [SWS_CORE_00345] and [SWS_CORE_00346]), `SetError()` (see [SWS_CORE_00353] and [SWS_CORE_00354]), or `SetResult()` (see [SWS_CORE_00355] and [SWS_CORE_00356]) on the `ara::core::Promise` which is created by the Method implementation (skeleton application code), lead to undefined behavior.⌋

7.5.4 Query Service Event Subscription State on Skeleton side

[SWS_CM_12012] Subscription State change handler ⌈The handler SubscriptionStateChangeHandler defined in [SWS_CM_00311], [SWS_CM_12008] and [SWS_CM_12009] shall be called by the Communication Management implementation as soon as the subscription state of this event has changed. Handler may be overwritten during runtime.⌋

[SWS_CM_12013] Call SubscriptionStateChangeHandler on Skeleton side with kSubscribed ⌈The Communication Management shall call the SubscriptionStateChangeHandler on the skeleton side with the value kSubscribed whenever the number of active subscriptions to this event become more than 0.⌋

[SWS_CM_12014] Call SubscriptionStateChangeHandler on Skeleton side with kNotSubscribed ⌈The Communication Management shall call the SubscriptionStateChangeHandler on the skeleton side with the value kNotSubscribed whenever the number of active subscriptions to this event become 0.⌋

[SWS_CM_12015] Query Subscription State on Skeleton side ⌈GetSubscriptionState on the skeleton side shall return kSubscribed if at least one active subscription to this event exists and kNotSubscribed otherwise. kSubscriptionPending shall not be used on the Server side.⌋

7.5.5 Send event

For the event sending C++ API reference, see chapter 8.3.2.2.2.1, 8.3.2.2.2.2 and 8.3.2.2.1.1.

To support sending of events where the data is owned by the application and continuously updated and the data is explicitly created for sending, the `Send()` / `Send()` method shall be provided in two ways: One where the application is owner of the data and the `Send()` / `Send()` method makes a copy for sending and one where Communication Management is responsible for the data and the application is not allowed to do anything with the data after sending.

[SWS_CM_99031] Send event where application is responsible for the data

Upstream requirements: RS_CM_00201

⌈As defined in [SWS_CM_00162], the `Send()` / `Send()` method of the specific `Event` class where the application is responsible for the data and the Communication Management creates a copy for sending shall be used whenever the application wants to work further with the data.⌋

[SWS_CM_99032] Send event where Communication Management is responsible for the data

Upstream requirements: RS_CM_00201

⌈As defined in [SWS_CM_90437], the `Send()` / `Send()` method of the specific `Event` class where the Communication Management is responsible for the data and the application is not allowed to access the data after sending shall be used whenever the data is created explicitly for sending and no further processing is happening afterward by the application itself.

Before sending the event, the corresponding data has to be requested from the Communication Management (see [SWS_CM_99033]) and filled with the respective data.⌋

[SWS_CM_99033] Allocating data for event transfer

Upstream requirements: RS_CM_00201

⌈Data shall be requested by calling the `Allocate()` method of the specific `Event` class as defined in [SWS_CM_90438]. By calling the `Send()` / `Send()` method with the data, it is ensured that the data will be freed by the Communication Management.⌋

NOTE! Since the `SampleAllocateePtr` pointer type behaves like a `std::unique_ptr` during the lifetime of the skeleton where it is allocated, the ownership of the pointer has to be transferred via `std::move` for utilizing zero-copy optimizations.

[SWS_CM_00086] Precondition for Skeleton destruction ⌈To avoid dangling references to `Event` sample data (referred to by a `SampleAllocateePtr`) which has not yet been handed back to communication management by the application, it is required that all `Event` sample data that has been allocated with the `Allocate` method, is either

- sent (using the `Send` method defined in [SWS_CM_90437])
- released by calling either
  - `SampleAllocateePtr::operator=(nullptr)`
  - `SampleAllocateePtr::reset()`
- destroyed by
  - explicitly calling `SampleAllocateePtr::~SampleAllocateePtr()`
  - implicitly triggering `SampleAllocateePtr::~SampleAllocateePtr()` (e.g., by letting it go out of scope)

before destroying the `Skeleton` (e.g., by letting it go out of scope). If this precondition is not met, a violation shall be raised as per [SWS_CORE_00003].⌋

7.5.6 Processing of service methods

For the processing of service methods C++ API reference, see chapter 8.3.5.2.6.1.

The *Method Call Processing Mode*, defined in [SWS_CM_00301] allows the implementation providing the service method to select how the incoming service method invocations are processed. The selection is valid for all the methods of the specific `ServiceSkeleton` instance. The *Method Call Processing Mode* is set as a parameter of the `ServiceSkeleton` constructor defined by [SWS_CM_00130].

[SWS_CM_10411] Service method processing modes

Upstream requirements: RS_CM_00211

⌈The following service method processing modes shall be supported:

- **Polling**: Instead of calling a provided service method, the Communication Management software collects incoming service method invocations. The processing of each invocation is explicitly triggered by the implementation providing the service method using the mechanism defined in [SWS_CM_00199].
- **Event-driven, concurrent**: The Communication Management software activates the invoked service method when the invocation arrives. Consumer concurrent calls are allowed and will be processed concurrently on provider side by using different threads.
  This is the default mode.
- **Event-driven, sequential**: The Communication Management software activates the invoked service method when the invocation arrives. Consumer concurrent calls are allowed, but will not be processed concurrently on provider side, by instead executing them one after the other to avoid the need of synchronization mechanisms in the implementation providing the service method.

⌋

The `ProcessNextMethodCall()` defined in [SWS_CM_00199] allows the implementation providing the service method to trigger the execution of the next service consumer method call at a specific point of time if the processing mode is set to `Polling`.

7.5.7 Registering handler with null pointer

[SWS_CM_10417] Register handlers failure due to null pointer or empty function.

Upstream requirements: RS_CM_00217, RS_CM_00218, RS_CM_00219

⌈Passing a `nullptr` or an empty `std::function` as argument to the following ara::com methods is not allowed and shall be handled as violation :

- `RegisterGetHandler` ([SWS_CM_00114], [SWS_CM_11360])

- `RegisterSetHandler` ([SWS_CM_00116], [SWS_CM_11362])
- `SetSubscriptionStateChangeHandler`([SWS_CM_00333], [SWS_CM_11354], [SWS_CM_12008], [SWS_CM_12009])
- `SetReceiveHandler`([SWS_CM_00181], [SWS_CM_11356], [SWS_CM_00250], [SWS_CM_00352])
- `StartFindService` ([SWS_CM_00123], [SWS_CM_00623])
- `SetServiceStateChangeHandler` ([SWS_CM_01074], [SWS_CM_01076])

⌋

7.5.8 Registering get handlers for fields

For the registering get handlers for fields C++ API reference, see chapter 8.3.3.2.3.2 and 8.3.3.2.3.1.

[SWS_CM_10412] Invoking GetHandlers

Upstream requirements: RS_CM_00218

⌈The registered `GetHandler` shall be called by the implementation whenever the Communication Management receives a `Get()`.⌋

7.5.9 Registering set handlers for fields

For the registering set handlers for fields C++ API reference, see chapter 8.3.3.2.3.4, 8.3.3.2.3.3 and 8.3.3.2.2.1.

[SWS_CM_10413] Invoking SetHandlers

Upstream requirements: RS_CM_00218

⌈The registered `SetHandler` shall be called by the implementation whenever the Communication Management receives a `Set()`.⌋

Note: Upon a call to the `SetHandler`, the Service Provider has to validate the received `field` value (it can accept, modify or reject it). After that, it sets the new value in the future object (see [SWS_CM_00116]). If the `SetHandler` needs to access the current `field` value to validate the new `field` value, the skeleton implementation has to provide a replica of the underlying `field` value that is accessible from application level.

[SWS_CM_10415] Notify the Field value after a call to the SetHandler function

Upstream requirements: RS_CM_00218

⌈The Communication Management implementation shall take the effective `field` value returned by the `SetHandler` function, and send it back to the requester as return value of the `Set()` function (see [SWS_CM_00113]), and to all the other subscribed entities via notification (see [SWS_CM_00119]).⌋

[SWS_CM_00128] Ensuring the existence of valid Field values

Upstream requirements: RS_CM_00218

⌈To ensure the existence of a valid field values upon a call to the `Subscribe()` method (see [SWS_CM_00141]) or to the `Get()` method (see [SWS_CM_00112]) the ara::com implementation shall do the following: If a service containing a `Field` is offered via a call to `OfferService()` (see [SWS_CM_00101]), if `Update()` has not been called yet and one or more of the following applies:

- `hasNotifier` = `true`
- `hasGetter` = `true` and a `GetHandler` (see [SWS_CM_00114]) has not yet been registered.

Then the error code `ComErrc::kFieldValueNotInitialized` shall be returned in the result type of `OfferService()`. The error shall be logged.⌋

[SWS_CM_00129] Ensuring the existence of SetHandler

Upstream requirements: RS_CM_00218

⌈Upon a call to `OfferService()` in a skeleton implementation for a given service, the following error check shall be made: if for at least one contained `Field` having `hasSetter` = `true` no `SetHandler` (see [SWS_CM_00116]) has been registered yet, the error code `ComErrc::kFieldSetHandlerNotSet` shall be returned in the result type of `OfferService()`. The error shall be logged.⌋

7.5.10 Find service

For the find service C++ API reference, see chapter 8.2.8.2.4.1, 8.2.8.2.4.2, 8.2.8.2.4.10, 8.2.8.2.4.9, 8.2.8.2.4.7, 8.2.8.2.4.8 and 8.2.8.2.4.11.

[SWS_CM_00124] Find service handler invocation

Upstream requirements: RS_CM_00102

⌈After calling the `StartFindService` method, the `ara::com::FindServiceHandler` shall be called by the Communication Management software to receive the found services. By the first call, the `ara::com::FindServiceHandler` shall receive the

initially known matches, if there are any. In following, the `ara::com::FindServiceHandler` shall be called every time the availability of any of the services matching the given instance criteria changes.⌋

[SWS_CM_10382] Calling stop find service for already stopped finds

Upstream requirements: RS_CM_00102

⌈Calls to the `StopFindService()` method using a `ara::com::FindServiceHandle` obtained from a `StartFindService` that already has been stopped shall be silently ignored.⌋

7.5.11 Service proxy creation

For the service proxy creation C++ API reference, see chapter 8.2.8.2.2.1, 8.2.8.2.3.1, 8.2.8.2.4.3, 8.2.8.2.1.3, 8.2.8.2.1.4, 8.2.8.2.1.1, 8.2.8.2.1.2 and 8.2.8.2.1.5.

[SWS_CM_10491] Re-establishing service connection

Upstream requirements: RS_CM_00102, RS_CM_00107

⌈In case the service becomes temporarily unavailable (due to restart, network problem or so), or if an error occurs while establishing a connection to the service, the error shall be logged, and the Communication Management shall retry to establish the connection once the next offer is received.⌋

7.5.12 Service proxy destruction

[SWS_CM_10446] Destruction of service proxy

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00127, RS_AP_00145

⌈The destructor of each specific `ServiceProxy` class shall destroy the `ara::core::Promise` instances corresponding to the `ara::core::Future` instances returned by `operator()` of the respective `Method` (see [SWS_CM_00196]) or by the `Get()` or `Set()` method of the respective `Field` class (see [SWS_CM_00112] and [SWS_CM_00113]) by explicitly or implicitly invoking the destructor of the `ara::core::Promise` (see [SWS_CORE_00349]). This in turn will make the corresponding `ara::core::Future` ready (if this is not already the case) with an `ara::core::ErrorCode` (see [SWS_CORE_00501]) where the error domain is set to `ara::core::FutureErrorDomain` (see [SWS_CORE_00421]) and the value is set to `ara::core::future_errc::broken_promise` (see [SWS_CORE_00400]).⌋

[SWS_CM_00087] Precondition for Proxy destruction ⌈To avoid dangling references to `Event` sample data (referred to by a `SamplePtr`) which has not yet been

handed back to communication management by the application, it is required that all `SamplePtr`s provided by `GetNewSamples()` are released (by calling any of the APIs defined in [SWS_CM_11547], [SWS_CM_11539], [SWS_CM_11540], or [SWS_CM_-11545]) before destroying the `Proxy` (e.g., by letting it go out of scope). If this precondition is not met, a violation shall be raised as per [SWS_CORE_00003].⌋

7.5.13 Service event subscription

For the service event subscription C++ API reference, see chapter 8.2.2.2.4.4, 8.2.2.2.4.6, 8.2.2.2.4.1, 8.2.2.2.4.3, 8.2.2.2.4.2 and 8.2.2.2.4.5.

[SWS_CM_00700] Ensure memory allocation of maxSampleCount samples

Upstream requirements: RS_CM_00103

⌈The Communication Management shall ensure, that after returning from method `Subscribe()` sufficient memory resources are available, so that the number of samples given in parameter `maxSampleCount` can be concurrently accessed by application layer.⌋

[SWS_CM_99035] Subscription State change handler on the Proxy side

Upstream requirements: RS_CM_00106

⌈The handler `SubscriptionStateChangeHandler` defined in [SWS_CM_00311] shall be called for the Proxy side by the Communication Management implementation as soon as the subscription state of this event has changed. Handler may be overwritten during runtime.⌋

[SWS_CM_00313] Call SubscriptionStateChangeHandler with kSubscriptionPending on the Proxy side

Upstream requirements: RS_CM_00103, RS_CM_00104, RS_CM_00106, RS_CM_00107

⌈The Communication Management shall call the `SubscriptionStateChangeHandler` on the Proxy side with the value `kSubscriptionPending` in the following cases:

- the client subscribes to an event and the actual subscription does not happen immediately (e.g. due to a bus protocol)
- the client is subscribed to an event and Communication Management has detected that the server instance is currently not available (due to restart, network problem or so)

⌋

[SWS_CM_12006] Asynchronous nature of Subscribe()

Upstream requirements: RS_CM_00106, RS_CM_00107

⌈In order to keep application functionality robust against Network Binding configuration changes, applications shall assume asynchronous operation when calling `Subscribe()`.

This implies not assuming success in the subscription process until `GetSubscriptionState()` or the handler set by `SetSubscriptionStateChangeHandler()` have reported kSubscribed, even if `Subscribe()` has returned with no error.

⌋

[SWS_CM_00085] Precondition for Event/Field unsubscription ⌈To avoid dangling references to `Event` sample data (referred to by a `SamplePtr`) which has not yet been handed back to communication management by the application, it is required that all `SamplePtr`s provided by `GetNewSamples()` are released (by calling any of the APIs defined in [SWS_CM_11547], [SWS_CM_11539], [SWS_CM_11540], or [SWS_CM_11545]) before before calling `Unsubscribe()`. If this precondition is not met, a violation shall be raised as per [SWS_CORE_00003].

⌋

[SWS_CM_00314] Call SubscriptionStateChangeHandler with kSubscribed on the Proxy side

Upstream requirements: RS_CM_00103, RS_CM_00104, RS_CM_00106, RS_CM_00107

⌈The Communication Management shall call the `SubscriptionStateChangeHandler` on the Proxy side with the value `kSubscribed` in the following cases:

- the client subscribes to an event and the actual subscription is established successfully
- the client is subscribed to an event and the actual subscription is re-established again after being temporarily unavailable (due to restart, network problem or so)

⌋

[SWS_CM_00315] Re-establishing an active subscription

Upstream requirements: RS_CM_00103, RS_CM_00104, RS_CM_00106, RS_CM_00107

⌈The Communication Management shall re-establish the actual subscription again after the server service being temporarily unavailable (due to restart, network problem or so). This shall work independently of whether a network binding is involved or not. The re-establishment shall also provide a possible update of binding specific connection properties if needed.⌋

7.5.14 Receive event

For the event data access C++ API reference, see chapter 8.2.2.2.3.2 and 8.2.2.2.3.1.

[SWS_CM_00703] Sequence of actions in GetNewSamples

Upstream requirements: RS_CM_00202

⌈In the context of the `GetNewSamples()` call, the Communication Management shall do the following steps repeatedly:

- get next received event data sample from underlying receive buffers.
- de-serialize the data, if needed.
- place the de-serialized data sample of type `SampleType` in the local cache.
- call user provided `f` with a `ara::com::SamplePtr` (including `ara::com::e2e::ProfileCheckStatus`) referencing the data sample located in local cache.

until at least one of the following conditions is true:

- `maxNumberOfSamples` have already been fetched from the underlying receive buffers within this `GetNewSamples()` call.
- `maxSampleCount` reached. I.e. the application is currently holding exactly as many `ara::com::SamplePtr`s provided by this `Event` class instance, than it has committed in call to `Subscribe()` via `maxSampleCount`.
- no new data samples available from underlying receive buffers.

⌋

[SWS_CM_00707] Calculation of Free Sample Count

Upstream requirements: RS_CM_00202

⌈

- After call to `Subscribe()` with parameter `maxSampleCount` set to N and *before* any call to `GetNewSamples()` on the same Event class instance, a call to `GetFreeSampleCount()` shall return N.
- Each `ara::com::SamplePtr` created by the Communication Middleware in the context of a call to `GetNewSamples()` on the same `Event` class instance shall lead to a decrement of count of free samples.
- Each destruction or `std::nullptr_t` assignment (see [SWS_CM_00306]) of a `ara::com::SamplePtr` instance created from this `Event` class instance shall lead to an increment of count of free samples.

⌋

[SWS_CM_00709] FIFO semantics

Upstream requirements: RS_CM_00203

⌈The Communication Management shall provide buffering with FIFO semantics between sender and receiver of events.⌋

Note: The Implementation of such a FIFO buffer (i.e. whether to be in Kernel Space, Shared Memory Space or IPC-Daemon Space) was not further detailed on purpose.

[SWS_CM_12007] New data samples received by CM at execution time of receive handler

Upstream requirements: RS_CM_00203

⌈In case new data samples arrive at Communication Management side during the execution of a user defined receive handler, Communication Management shall postpone the next call to receive handler until the previous call to receive handler is finished.⌋

[SWS_CM_00710] No implicit context switches

Upstream requirements: RS_CM_00203

⌈When no `ReceiveHandler` has been set at the proxy side via `SetReceiverHandler()`, new SampleData shall only be received by directly invoking `GetNewSamples()` (polling behavior). Reception of new events itself shall not lead to an implicit context switch in the local receiver process (i.e. if only polling behavior is used). In case a `SetReceiveHandler()` is enabled, a context switch shall be enforced with the reception of new events to schedule/invoke the `ReceiveHandler`.⌋

7.5.14.1 Receive event by polling

For the polling access no additional APIs on top of 8.2.2.2.3.2 and 8.2.2.2.3.1 are needed.

7.5.14.2 Receive event by getting triggered

For the receive event by getting triggered C++ API reference, see chapter 8.2.2.2.5.1.

[SWS_CM_00182] Event Receive Handler call serialization

Upstream requirements: RS_CM_00203

⌈The Communication Management shall serialize calls to the registered `EventReceiveHandler` function as it is not guaranteed that the callback function is reentrant.⌋

[SWS_CM_00711] `GetNewSamples()` shall provide data samples if `GetFreeSampleCount()` is not 0

Upstream requirements: RS_CM_00203

⌈After the Communication Management has called the registered `EventReceiveHandler` function for a specific `Event` class instance, the next call to `GetNewSamples()` on the same instance shall provide at least one data sample as long as `GetFreeSampleCount()` is not already returning 0 at the point in time of the call.⌋

7.5.15 Service trigger subscription

For the service trigger subscription C++ API reference, see chapter 8.2.7.1.3.1 and 8.2.7.1.3.2.

Getting subscription state and set a subscription change handler for `Trigger` is the same as for `Event`. The following specification are also valid for `Trigger`:

- [SWS_CM_00316] Query Subscription State.
- [SWS_CM_00333] Set Subscription State change handler.
- [SWS_CM_11354] Execution Context for setting Subscription State change handler.
- [SWS_CM_00334] Unset Subscription State change handler.
- [SWS_CM_00313] Call SubscriptionStateChangeHandler with kSubscriptionPending.
- [SWS_CM_00314] Call SubscriptionStateChangeHandler with kSubscribed.
- [SWS_CM_00315] Re-establishing an active subscription.

7.5.16 Receive trigger

For the trigger data access C++ API reference, see chapter 8.2.7.1.2.1.

[SWS_CM_00227] Sequence of actions in `GetNewTriggers()`

Upstream requirements: RS_CM_00202

⌈In the context of the `GetNewTriggers()` (see [SWS_CM_00226]) call, the Communication Management shall get the number of triggers occurred since the last call of `GetNewTriggers()`.⌋

7.5.16.1 Receive trigger by getting triggered

For the receive event by getting triggered C++ API reference, see chapter 8.2.7.1.4.2, 8.2.7.1.4.1 and 8.2.7.1.4.3.

The following specification are also valid for `Trigger`

- [SWS_CM_00183] Disable service event trigger

7.5.17 Call a service method

For the call a service method C++ API reference, see chapter 8.2.6, 8.2.5.1.1.1 and 8.2.4.1.1.1.

[SWS_CM_10414] Initiate a method call

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_AP_00138

⌈At the point of time when the caller calls the method (see [SWS_CM_00196]), the Communication Management software does not know yet if the result shall be returned with synchronous or asynchronous behavior. Therefore the Communication Management software shall instantiate the `ara::core::Future` object to be returned to the caller, but shall not perform actions which lead to uncontrolled context switches from the caller point of view, e.g. an asynchronous event-style mechanism for a wait-on-event.⌋

[SWS_CM_10371] Context of return checked errors

Upstream requirements: RS_CM_00211, RS_CM_00212, RS_CM_00213, RS_CM_00214

⌈If during processing of a method call one of the checked errors occurs, the corresponding `ara::core::ErrorCode` shall be returned in the context of the `GetResult()/get()` call.⌋

See Section 7.5.20.6) for the definition of checked errors.

[SWS_CM_90436] No checked errors for `Fire and Forget method` calls

Upstream requirements: RS_CM_00225

⌈There shall be no checked errors returned for `Fire and Forget method` calls.⌋

[SWS_CM_00194] Cancel the method call

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_AP_00114, RS_AP_00127

⌈The `destructor` of the returned `ara::core::Future` object shall be used by the caller to cancel the request after issuing a method call. Deleting the returned `ara::core::Future` object shall result in the abort of the method call and ensure that any related buffers are released and no result is returned to the caller.⌋

This is a mechanism on client side to tell the Communication Management software that the caller is not interested in the method result anymore. Propagating the cancellation of a method call to the skeleton side, which implements the service interface method is network binding dependant and therefore purposely not in scope of Communication Management specification and left as optional.

[SWS_CM_00195] Retrieving results of the method call

Upstream requirements: RS_CM_00212, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127, RS_AP_00139

⌈The method `GetResult()` of the returned `ara::core::Future` object shall be used to retrieve the result of the method call as `ara::core::Result`. The call of the method `GetResult()` will block if there is not yet a result available and will return after the result has been received returning an object of the respective `Output` or an error. As an alternative, `get()` returns the contained object of the result from `GetResult()`, or throws the contained error as exception, respectively.⌋

[SWS_CM_00192] Synchronous behavior of method call

Upstream requirements: RS_CM_00212, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127

⌈To achieve synchronous behavior of the method call, the methods of `ara::core::Future` object with blocking behavior shall be used because they only return when the output of the method call according to [SWS_CM_00196] is available: `get()`, `wait()`, `wait_for()` and `wait_until()`. With the call of one of these methods and the result still pending, the Communication Management software is allowed to perform actions which lead to uncontrolled context switches from the caller point of view, e.g. an asynchronous event-style mechanism for a wait-on-event.⌋

Note that there are situations where the methods of an `ara::core::Future` object with blocking behavior will block forever. The adaptive application will need to gracefully handle such a situation. Prominent examples for such situations are the following ones:

- the request message or the response message of the (remote) service method call gets lost
- the implementation for the service method in the subclass of the respective `ServiceSkeleton` (see [SWS_CM_00194]) does not return (i.e., hangs)

`ara::com` will **not** internally perform some kind of timeout supervision in order to eventually unblock those blocking `ara::core::Future` methods. If such a timeout supervision is desired from the perspective of the adaptive application, it is up to the adaptive application to implement according mechanisms, e.g., by using the `wait_for()` and `wait_until()`, or the `is_ready()` methods of the `ara::core::Future`.

On the other hand there are situations where the `ara::com` implementation on the client side **knows** that an issued (remote) service method call will not succeed and

thus would block forever. Prominent examples for such situations are the following ones:

- the sending of request message of the (remote) service method failed locally (i.e., the corresponding system or library call indicated an error)
- the received response message partly contains malformed message content but contains sufficient correct information allowing to determine the method this response is targeted at (i.e., there is sufficient information available about who to notify/which `ara::core::Future` to fulfill) – in case of the SOME/IP network binding (see Section 7.2.1) this would be a response message where
  - the layer 2 and layer 4 checksums are correct
  - the SOME/IP header (which contains the method ID) is intact (e.g., in case of a SOME/IP response message, the checks described in [SWS_CM_10313] are passed)
  - the de-serialization of the payload fails though

[SWS_CM_10440] Aborting method calls in case of locally detected platform failures

Upstream requirements: RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127

⌈To notify the adaptive application about locally detected platform failures which prevent an issued (remote) service method call from succeeding, the `ara::com` implementation shall make the `ara::core::Future` returned by `operator()` of the respective `Method` class (see [SWS_CM_00196]) or by the `Get()` or `Set()` method of the respective `Field` class (see [SWS_CM_00112] and [SWS_CM_00113]) ready by invoking the `SetError` (see [SWS_CORE_00353]) operation of the `ara::core::Promise` corresponding to this `ara::core::Future` with an `ara::core::ErrorCode` (see [SWS_CORE_00501]) where the error domain is set to `ara::com::ComErrorDomain` (see [SWS_CM_11329]) and the value is set to `kNetworkBindingFailure` (see [SWS_CM_10432]) as an argument.⌋

[SWS_CM_00048] Aborting method calls in case of detected unconfigured application error

Upstream requirements: RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127

⌈To notify the adaptive application about a detected unconfigured application error which is returned from an issued (remote) service method call, the `ara::com` implementation shall make the `ara::core::Future` returned by the function call `operator()` of the respective `Method` class (see [SWS_CM_00196]) ready by invoking the `SetError` (see [SWS_CORE_00353]) operation of the `Promise` corresponding to this `Future` with an `ara::core::ErrorCode` (see [SWS_CORE_00501]) where the error domain is set to `ara::com::ComErrorDomain` (see [SWS_CM_11329]) and the value is set to `kUnknownApplicationError` (see [SWS_CM_10432]) as an argument.⌋

[SWS_CM_00049] Aborting method calls in case of detected unspecified E2E error

Upstream requirements: RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127

⌈To notify the adaptive application about a detected unspecified E2E error which is returned from an issued (remote) service method call, the `ara::com` implementation shall make the `ara::core::Future` returned by the function call `operator()` of the respective `Method` class (see [SWS_CM_00196]) ready by invoking the `SetError` (see [SWS_CORE_00353]) operation of the `Promise` corresponding to this `Future` with an `ara::core::ErrorCode` (see [SWS_CORE_00501]) where the error domain is set to `ara::com::e2e::ComE2EErrorDomain` (see [SWS_CM_12503]) and the value is set to `kUnspecifiedE2EError` (see [SWS_CM_10474]) as an argument.⌋

[SWS_CM_00193] Asynchronous behavior of method call with polling

Upstream requirements: RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00127

⌈To achieve asynchronous behavior of the method call with polling on the result availability, the non-blocking method `is_ready()` of `ara::core::Future` object shall be used. If `is_ready()` returns `true`, the next call of `Get()` shall not block, but immediately return the valid value.⌋

Note:
When the user just calls `is_ready()` of `ara::core::Future` and on positive response, finally `GetResult()`/`get()` of `ara::core::Future`, retrieving the result works polling-based without any overhead in the middleware and uncontrolled context switches due to asynchronous event-style mechanisms.

[SWS_CM_00197] Asynchronous behavior of method call with notification

Upstream requirements: RS_CM_00213, RS_CM_00215, RS_AP_00114, RS_AP_00127, RS_AP_00138

⌈To achieve asynchronous behavior of the method call with event-driven notification on the result availability, the non-blocking method `then()` of `ara::core::Future` object shall be used. It allows to register a function, which gets asynchronously called in case the future has a valid result.⌋

Note:
In order to report an `ApApplicationError` (according to the interface description in the Manifest), the implementation of a service method stores the corresponding `ara::core::ErrorCode` representing this `ApApplicationError` (see [30]) into the `ara::core::Promise` object, from which the `ara::core::Future` is returned to the caller of the service method implementation. This `ara::core::ErrorCode` will then be used by the respective network binding to include the error information in the method response message (see e.g., [SWS_CM_10308] and [SWS_CM_10428]

for the SOME/IP network binding or [SWS_CM_11102] and [SWS_CM_10431] for the DDS network binding).

7.5.18 Update notification events for fields

[SWS_CM_00120] Provision of an update notification event for a Field

Upstream requirements: RS_CM_00218

⌈If `hasNotifier` is true, update notification events for the `Field` shall be provided as of the following requirements:

- [SWS_CM_00141] Method to subscribe to a service event. This subscribe leads immediately to a service event that contains the initial field value send from provider side to the consumer.
- [SWS_CM_00151] Method to unsubscribe from a service event.
- [SWS_CM_00316] Method to query the subscription state.
- [SWS_CM_00701] Method to receive a service event using polling.
- [SWS_CM_00181] Method to enable service event trigger.
- [SWS_CM_00182] Event Receive Handler call serialization.
- [SWS_CM_00183] Method to disable service event trigger.
- [SWS_CM_00333] Method to set a subscription state change handler.
- [SWS_CM_00334] Method to unset a subscription state change handler.

Except that the corresponding methods reside in the `Field` class instead of the `Event` class.⌋

7.5.19 Instance Specifier Translation

For the instance specifier translation C++ API reference, see chapter 8.5.2.1.1.

[SWS_CM_10452] `ara::core::InstanceSpecifier` translation to `ara::com::InstanceIdentifier`s

Upstream requirements: RS_CM_00200, RS_AP_00137

⌈The Communication Management shall translate an `InstancSpecifier` to `ara::com::InstanceIdentifier`s. Based on the match there shall be zero, 1 or multiple `ara::com::InstanceIdentifier`s .⌋

7.5.20 API Data Types

This chapter describes the functionality of the data types used by the ara::com API, both the specific ones which are part of the standardized proxy and skeleton interfaces, and the ones derived from the description based on the AUTOSAR meta-model.

7.5.20.1 Service Identifier Data Types

For the Service Identifier Data Types C++ API reference, see chapter 8.9.1.

The data types described in this chapter are used to identify a specific service or service instance.

There might exist different instances of exactly the same service in the system. To handle this, an `ara::com::InstanceIdentifier` or an `ara::core::InstanceSpecifier` are used to identify a specific instance of a service.

An `ara::com::InstanceIdentifier` (see [SWS_CM_00302]) is a unique identifier of a specific instance of a service, needed to distinguish different instances of exactly the same service in the system. It contains instance information and information about the service type. This will make the `ara::com::InstanceIdentifier` unique for different instances.

A service can be identified at least by a fully qualified name and a version.

[SWS_CM_00088] Service Identifier String ⌈The format of the string passed to the constructor (see [SWS_CM_11520]), or returned by the ToString() method (see [SWS_CM_11522]) is specific to the Communication Management software provider, and not standardized.⌋

[SWS_CM_99029] Service Contract Version

Upstream requirements: RS_CM_00500

⌈The value of the service contract major version (`serviceContractVersionMajor`) shall be derived from the `majorVersion` attribute in the `ServiceInterface`. The value of the service contract minor version (`serviceContractVersionMinor`) shall be derived from the `minorVersion` attribute in the `ServiceInterface`.⌋

The following data types are used for the handling of services on the service consumer side.

[SWS_CM_99030] Find Service Handle

Upstream requirements: RS_CM_00102, RS_AP_00122, RS_AP_00119

⌈To identify a triggered request to find a service, the `StartFindService(InstanceIdentifier)` method of [SWS_CM_00123] shall return a `ara::com::FindServiceHandle` which is used as parameter to cancel this request with `StopFindService()` as described in [SWS_CM_00125].⌋

The usage of the API to find service instances, as defined in [SWS_CM_00122] and [SWS_CM_00123], provides a *handle container* (see [SWS_CM_00319]) holding a list of *handle*s. Each *handle* represents an existing service instance and by passing the *handle* as parameter to the proxy constructor [SWS_CM_00131], it allows the ara::com API user to create a proxy instance to access this service instance.

7.5.20.2 Event Related Data Types

For the Event Related Data Types C++ API reference, see chapter 8.8.1, 8.4.1.6, 8.4.1.1 8.4.1.10 and 8.4.1.11.

Event handling on receiver side is based on queued communication with configurable cache sizes. The cache size for a specific event of a proxy instance is determined by the Communication Management, when subscribing to a specific event by [SWS_CM_00141].

After the receiver subscribed to an event, the method `GetNewSamples`, as defined in [SWS_CM_00701], is used to retrieve the *data samples* of that event. In the context of `GetNewSamples` application provided callback functions are called by the Communication Management, where *Sample Pointer*s to the data samples retrieved from underlying queues are passed in. A *Sample Pointer* (see [SWS_CM_00306]) is an alias for an event data type pointer.

On the event provider side, it is possible to let the Communication Management allocate the memory for the storage of the data before sending it as defined in [SWS_CM_90438]. A `ara::com::SampleAllocateePtr` (see [SWS_CM_00308]) is an alias for an event data type pointer used both for allocation and data sending.

The event receiver can register an *Event Receive Handler* (see [SWS_CM_00309]) as a callback to get notified if new event data has arrived. The callback function itself is defined in the event consumer implementation; the *Event Receive Handler* type is just an general purpose function alias for the use in the method `SetReceiveHandler()` as defined by [SWS_CM_00181].

The event receiver can monitor the state of a service event subscription by requesting or getting a notification of the *Subscription State* (see [SWS_CM_00310], [SWS_CM_00316] and [SWS_CM_00311]), as the real process of subscription might happen at a later point in time than the return of the call to `Subscribe`. The *Subscription State* related ara::com API methods require the definitions of a *Subscription State*

enumeration ([SWS_CM_00310]) and a *Subscription State Changed Handler* function wrapper.

7.5.20.3 Trigger Related Data Types

For the Trigger Related Data Types C++ API reference, see chapter 8.4.1.12.

The trigger receiver can register a *Trigger Receive Handler* (see [SWS_CM_00351]) as a callback to get notified if new trigger has arrived. The callback function itself is defined in the trigger consumer implementation; the *Trigger Receive Handler* type is just an general purpose function alias for the use in the method `SetReceiveHandler()` as defined by [SWS_CM_00250].

The trigger receiver can monitor the state of a service trigger subscription by requesting or getting a notification of the *Subscription State* (see [SWS_CM_00310], [SWS_CM_00316] and [SWS_CM_00311]), as the real process of subscription might happen at a later point in time than the return of the call to `Subscribe`. The *Subscription State* related ara::com API methods require the definitions of a *Subscription State* enumeration ([SWS_CM_00310]) and a *Subscription State Changed Handler* function wrapper.

The [SWS_CM_00310] and [SWS_CM_00311] are also valid for triggers as well.

7.5.20.4 Method Related Data Types

For the Method Related Data Types C++ API reference, see chapter 8.4.1.5.

Service method invocation on provider side can be executed in different processing modes, where the *Method Call Processing Mode* (see [SWS_CM_00301]) is set as a parameter of the `ServiceSkeleton` constructor defined by [SWS_CM_00130].

The expected behavior of each processing mode is described in [SWS_CM_00301].

7.5.20.5 Generic Data Types

[SWS_CM_10453] Implementation of `invalidValue`

Upstream requirements: RS_CM_00001

⌈For AUTOSAR data types which have an `invalidValue` specified, header file shall also contain the following definition in the same namespace as type declaration:

```
constexpr static <SourceDataType> kInvalidValue<DataType> = <InvalidValue>;
```

where

- `<DataType>` is the short name of the data type

- <SourceDataType> is data type, implicitly convertible to <DataType>;
  In simplest case <DataType> itself.
- <InvalidValue> is the value defined as invalidValue for the data type

⌋

Note: invalidValues are only applicable to CppImplementationDataType of category TYPE_REFERENCE and STRING.

7.5.20.6 Error Related Data Types

[SWS_CM_11265] Use of general ara::com errors

Upstream requirements: RS_CM_00211, RS_AP_00119

⌈Any Error as per [SWS_CORE_00020] of a ServiceInterface shall be reported via the return type as specified in [3].⌋

In ara::com, there are the following types of Errors:

1. General ara::com errors: These errors can occur in a call of a ServiceInterface method but are not specific to a certain ServiceInterface. They are defined in the error domain ara::com::ComErrorDomain.
2. E2E errors: These errors are specific to E2E checks. They are defined in the error domain ara::com::e2e::ComE2EErrorDomain (see chapter 7.5.20.7)
3. Application Errors: These errors are specific to a certain ServiceInterface call. They are defined as ApApplicationError in the meta-model.

7.5.20.7 E2E Related Data Types

Some data types are used only in context of e2e-protected communication of events.

[SWS_CM_12021] Mapping of `ara::com::e2e::ProfileCheckStatus`

Upstream requirements: RS_E2E_08534, RS_AP_00114, RS_AP_00115, RS_AP_00119

⌈

Enumeration literal: `ara::com::e2e::ProfileCheckStatus`	Profile independent result of `E2E_Check()`
`kOk`	OK
`kRepeated`	REPEATED
`kWrongSequence`	WRONGSEQUENCE
`kError`	WRONGCRC
n/a	NONEWDATA
`kCheckDisabled`	n/a

The E2E profile independent results according to [PRS_E2E_00677] shall be mapped to `ara::com::e2e::ProfileCheckStatus` according to this table.

⌋

The E2E state machine `ara::com::e2e::SMState` is determined by checking a history of `ara::com::e2e::ProfileCheckStatus`'. The current value of `ara::com::e2e::SMState` mirrors the current state of the E2E supervision, but is not necessarily applicable to all samples received during the last update.

[SWS_CM_12026] Mapping of `ara::com::e2e::ComE2EErrc`

Upstream requirements: RS_E2E_08534, RS_AP_00114, RS_AP_00115, RS_AP_00119

⌈

Enumeration literal of `ara::com::e2e::ComE2EErrc`	Profile independent result of `E2E_Check()`
`n/a`	OK
`kRepeated`	REPEATED
`kWrongSequence`	WRONGSEQUENCE
`kError`	WRONGCRC
`kNoNewData`	NONEWDATA

The E2E profile independent results according to [PRS_E2E_00677] shall be mapped to `ara::com::e2e::ComE2EErrc` according to this table.

⌋

[SWS_CM_12022] Mapping of `ara::com::e2e::SMState`

Upstream requirements: RS_E2E_08534, RS_AP_00114, RS_AP_00115, RS_AP_00119

⌈

Enumeration literal: `ara::com::e2e::SMState`	Communication channel status
`kValid`	VALID
`kNoData`	NODATA, DEINIT
`kInit`	INIT
`kInvalid`	INVALID
`kStateMDisabled`	n/a

The communication channel status according to [PRS_E2E_00678] shall be mapped to `ara::com::e2e::SMState` according to this table. `kIncompatibleQoS` may be reported by the DDS Middleware as described in chapter 2.2.4.1 "Communication Status" of [20]

⌋

7.5.21 API Header Files

The ara::com API C++ header files (see chapter 8) are either statically defined or generated by processing the ARXML configuration during the workflow:

- `ara::com Types Header File`
- `ara::com Runtime Header File`
- `Service Common Header File` (generated)
- `Service Skeleton Header File` (generated)
- `Service Proxy Header File` (generated)
- `Service Implementation Data Types Header Files` (generated, [30])
- `Service Method Application Errors Header Files` (generated, [30])

7.5.21.1 Service Implementation Data Types Header Files

The `Service Implementation Data Types Header Files` represent the language bound definitions of `CppImplementationDataType`s generated by a workflow tool from the `ServiceInterface`s. The processing rules of how the `ARA Language Binding Generator` shall create these, are specified in detail in [30]. The usage of these is described in [SWS_CM_01012].

7.5.21.2 Service Method Application Errors Header Files

The `Service Method Application Errors Header Files` are the language bindings of `ApApplicationError`s generated by a workflow tool from a `PortInterface`. The processing rules of how the `ARA Language Binding Generator` shall create these, are specified in [30].

7.6 Functional cluster lifecycle

The Communication Management functional cluster provides the primary communication infrastructure for Adaptive AUTOSAR, which is used by State Management. Because the interaction between State Management and Execution Management has a key impact on the lifecycle of the entire Adaptive AUTOSAR platform, the availability of communication infrastructure is essential for system state changes. AUTOSAR assumes the availability of this communication in the states Startup and Shutdown to the extent necessary to perform these states.

7.6.1 Startup

No special startup handling is needed for the Communication Management functional cluster. However, Communication Management provides the communication infrastructure used by State Management. Therefore, it is recommended to start Communication Management in parallel with Execution Management or after starting Execution Management but before starting State Management. Once State Management and Execution Management are operational, they should take control of the Communication Management lifecycle.

Please note that the specific implementation details and configuration of Language Binding, Communication Binding and Network Binding made by the integrator affects the specific requirements for a given deployment.

7.6.2 Shutdown

Control over this state of the Communication Management functional cluster lifecycle should be handled by State Management and Execution Management. However, Communication Management provides the communication infrastructure used by State Management. Therefore, the Communication Management functional cluster should maintain the functionality required by State Management as long as it is necessary.

Please note that the specific implementation details and configuration of Language Binding, Communication Binding, and Network Binding affects the shutdown strategy for a particular deployment. It is the responsibility of the system integrator to carefully consider when the Communication Management elements will terminate to ensure the

success of the system shutdown and notification of Applications. In particular the system integrator should provide a concept how to notify application processes still holding `ara::com::SamplePtr`'s to memory elements of communication management.

7.7 Reporting

7.7.1 Security Events

This functional cluster does not define any security events.

7.7.2 Log Messages

This functional cluster does not define any non-verbose log messages (i.e., modelled DLT messages).

7.7.3 Violation Messages

This section lists all violation messages (i.e., DLT messages logged for Violations according to [SWS_CORE_00021]) defined by this functional cluster.

Dlt-Message	InstanceSpecifierMappingIntegrityViolation		
Description	InstanceSpecifier either cannot be resolved in the model in the context of your executable, or it refers to a model element other than a PortPrototype. String format: "Violation detected in {processIdentifier} at {location}: Invalid InstanceSpecifer {instanceSpecifier} in a constructor of class: {className}"		
MessageId	0x80001ffc		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

Dlt-Message	PortInterfaceMappingViolation		
Description	The type of mapping does not match the expected type of PortInterface: {portInterfaceTypeName} referenced by a {mappingTypeName}. String format: "Violation detected in {processIdentifier} at {location}: Invalid InstanceSpecifer {instanceSpecifier} in a constructor of class: {className}"		
MessageId	0x80001ffb		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***

▽

△

processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

Dlt-Message	ProcessMappingViolation		
Description	Matching InstanceRef exists, but no matching (modelled) Process found that matches the (runtime) process. String format: "Violation detected in {processIdentifier} at {location}: Invalid InstanceSpecifer {instanceSpecifier} in a constructor of class: {className}"		
MessageId	0x80001ffa		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

Dlt-Message	InstanceSpecifierAlreadyInUseViolation		
Description	Violation message that is sent in case a constructor in the ara framework was called with an Instance Specifier already in use in this process. String format: "Violation detected in {processIdentifier} at {location}: InstanceSpecifer {instanceSpecifier} in constructor of class {className} already in use in this process"		
MessageId	0x80001ff9		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	NoUnit

Dlt-Message	InsufficientPermissionsViolation		
Description	Sent in case the caller had insufficient permissions for the requested operation. String format: "Violation detected in {processIdentifier} at {location} due to insufficient permissions: {message}"		
MessageId	0x80001fff		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit

▽

△

location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
message	Additional message that describes the cause of the access violation.	uint8 [encoding UTF-8]	NoUnit

[SWS_CM_12101] ViolationMessage ErroneousServiceHandleViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	ErroneousServiceHandleViolation		
Description	Violation message that is sent in case an erroneous service handle is passed to the Proxy constructor. String format: "Violation detected in {processIdentifier} at {location}: Erroneous service handle was passed to the {className} constructor"		
MessageId	0x80000ffe		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
processingMode	processingMode used to try to create the object.	uint8 [encoding UTF-8]	NoUnit
className	Name of the skeleton class that could not be constructed.	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_CM_12100] ViolationMessage WrongMethodCallProcessingModeViolation

Status: DRAFT
Upstream requirements: RS_AP_00142

⌈

Dlt-Message	WrongMethodCallProcessingModeViolation		
Description	Violation message that is sent in case a wrong processing mode is passed to the Skeleton constructor. String format: "Violation detected in {processIdentifier} at {location}: Wrong processing mode ({processingMode}) was passed to the {className} constructor"		
MessageId	0x80000fff		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
processIdentifier	Identifier of the process that caused the violation.	uint8 [encoding UTF-8]	NoUnit
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	NoUnit
processingMode	processingMode used to try to create the object.	uint8 [encoding UTF-8]	NoUnit

▽

△

className	Name of the skeleton class that could not be constructed.	uint8 [encoding UTF-8]	NoUnit

⌋

7.7.4 Production Errors

This functional cluster does not define any production errors (i.e., Diagnostic Events).

8 Communication API specification

This chapter provides a reference of the APIs defined by this functional cluster. The API is described in the following chapters in tables. Table 8.1 explains the content that is described in such an API table.

Kind:	Defines the kind of the declaration that this API table describes. The following values are supported: • class (Declaration of a class) • function (Declaration of a member or non-member function) • struct (Declaration of a structure) • type alias (Declaration of a type alias) • enumeration (Declaration of an enumeration) • variable (Declaration of a variable)	
Header File:	Defines the header file to be included according to [SWS_CORE_90001]	
Forwarding Header File:	Defines the forwarding header file to be included according to [SWS_CORE_90001]	
Scope:	Defines the scope that may be a namespace (in case of a class or non-member function) or a class declaration (in case of a member)	
Symbol:	Entity name	
Thread Safety:	Defines whether a function is thread-safe, not thread-safe, or conditional according to [SWS_CORE_13200] and [SWS_CORE_13202]	
Syntax:	Description of C++ syntax	
Template Param:	Template parameter (0..*)	Template parameter(s) used to parametrize the template
Parameters (in):	Parameter declaration (0..*)	Parameter(s) that are passed to the function
Parameters (out):	Parameter declaration (0..*)	Parameter(s) that are returned to the caller
Return Value:	Return type	Type of the value that the function returns
Exception Safety:	Defines whether a function is exception-safe, not exception safe or conditionally exception safe	
Exceptions:	List of exceptions that may be thrown from the function	
Violations:	List of violations that may occur in the function	
Errors:	Error type (0..*)	List of defined error codes that may be returned by the function with their recoverability class defined in [RS_AP_00160]. APIs can be extended with vendor-specific error codes. These are not part of the AUTOSAR SWS specifications
Description:	Brief description of the function	

Table 8.1: Explanation of an API table

8.1 Header: {<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h

[SWS_CM_01012] `Service Common Header File`: file name, includes and multiple inclusion guard

Upstream requirements: RS_CM_00001, RS_AP_00114, RS_AP_00116

⌈

Kind:	Header File	
Syntax:	`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h`	
Description:	For each modeled `ServiceInterface` a `Service Common Header File` is generated according to this directory path/file name convention. The `Service Common Header File` provides the common interface to a `Service Skeleton Header File` and `Service Proxy Header File`. It is implementation defined whether contained type definitions are directly defined in the `Service Common Header File` or split to further header files. The `Service Common Header File` shall: 1. Insert a multiple inclusion guard around the whole header file as per [SWS_CORE_90002] 2. Include the `ara::com Types Header File` as per [SWS_CM_01013] 3. Include each `Service Implementation Types Header File` used in the `ServiceInterface` as per [SWS_LBAP_00033] 4. Include the `ara::com Error Domain Header File` as per [SWS_CM_11329] 5. Include the `Service Method Application Errors Header Files` as per [30]	
Descriptors:	`{<si-namespace-derived-directory-path-lower>}`	as per [SWS_CM_01005] whereby: for each inner namespace in the hierarchy, an inner directory shall be created to contain the header file
	`{<si-shortname-lower>}`	`ServiceInterface`. `shortName` converted to lower-case.

Example:	`// File=n/n_plus_1/n_plus_2/si_common.h`
	`#ifndef N_NPLUS1_NPLUS2_SI_COMMON_H_ (1)`
	`#define N_NPLUS1_NPLUS2_SI_COMMON_H_ (1)`
	`#include "ara/com/types.h" (2)`
	`#include ".../path/to/impl_type_<type>.h" (3)`
	`#include "ara/com/com_error_domain.h" (4)`
	`#include ".../path/to/<aaed>_error_domain.h" (5)`
	`...`
	`#endif // N_NPLUS1_NPLUS2_SI_COMMON_H_ (2)`	

See also:	[SWS_CORE_90001], [SWS_CORE_90002], [SWS_LBAP_00033]	

⌋

8.1.1 Namespaces

8.1.1.1 {<hierarchical-namespace-list-lower-common>}

[SWS_CM_11500] Service Common Header File: service namespace

Upstream requirements: RS_AP_00114, RS_CM_00002

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h"
Scope:	--
Syntax:	namespace {<hierarchical-namespace-list-lower-common>}
Description:	The SymbolProps aggregated in the role PortInterface. namespace constructs the encapsulating C++ namespace hierarchy for the Service Header Files. For each namespace in the **ordered** list: namespace[N+1] shall be an inner namespace of namespace[N] converted to lower-case. Note: In order to avoid name clashes between Events, Fields, and Methods of different Service Interfaces in situations where the Events (ServiceInterface.event), Fields(ServiceInterface.field), and Methods(ServiceInterface.method) of the different ServiceInterfaces carry the same shortname, it is highly recommended to place different ServiceInterfaces into dedicated unique C++ namespaces. This is achieved by attaching the corresponding ordered SymbolProps to the ServiceInterfaces where the ordered SymbolProps differ in at least one of their symbol attributes.

Example:	```...
	namespace n {
	namespace n_plus_1 {
	namespace n_plus_2 { ... } // namespace n_plus_2
	} // namespace n_plus_1
	} // namespace n
	...```

See also:	[SWS_LBAP_00035]

⌋

8.1.1.2 {<hierarchical-namespace-list-lower-common>}::common

[SWS_CM_11501] Service Common Header File: common namespace

Upstream requirements: RS_AP_00114, RS_CM_00002

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h"
Scope:	namespace {<hierarchical-namespace-list-lower-common>}

▽

△

Syntax:	`namespace common`
Description:	Inner namespace for definitions common between a `Service Skeleton Header File` and `Service Proxy Header File`.

⌋

8.1.2 Class: {<si-shortname>}

[SWS_CM_01010] `Service Common Header File` Service Identifier and Service Contract Version

Upstream requirements: RS_AP_00114, RS_CM_00002

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common_fwd.h"	
Scope:	`namespace {<hierarchical-namespace-list-lower-common>}::common`	
Symbol:	`{<si-shortname>}`	
Syntax:	`class {<si-shortname>} {...};`	
Description:	Inner namespace for definitions common to a `Service Skeleton Header File` and `Service Proxy Header File`.	
Descriptors:	`{<si-shortname>}`	`ServiceInterface.shortName`
See also:	[SWS_CM_99029]	

⌋

8.1.2.1 Public Member Variables

8.1.2.1.1 serviceContractVersionMajor

[SWS_CM_11508] Definition of API variable {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}::serviceContractVersionMajor

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	variable
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h"
Scope:	class {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}
Symbol:	serviceContractVersionMajor
Type:	std::uint32_t
Syntax:	static std::uint32_t serviceContractVersionMajor;
Description:	Service Contract Major Version

⌋

8.1.2.1.2 serviceContractVersionMinor

[SWS_CM_11509] Definition of API variable {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}::serviceContractVersionMinor

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	variable
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h"
Scope:	class {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}
Symbol:	serviceContractVersionMinor
Type:	std::uint32_t
Syntax:	static std::uint32_t serviceContractVersionMinor;
Description:	Service Contract Minor Version

⌋

8.1.2.1.3 serviceIdentifier

[SWS_CM_11506] Definition of API variable {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}::serviceIdentifier

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	variable
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h"
Scope:	class {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}
Symbol:	serviceIdentifier
Type:	const ara::com::ServiceIdentifierType
Syntax:	static const ara::com::ServiceIdentifierType serviceIdentifier;
Description:	Service Identifier

⌋

8.1.2.1.4 serviceVersion

[SWS_CM_11507] Definition of API variable {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}::serviceVersion

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	variable
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_common.h"
Scope:	class {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}
Symbol:	serviceVersion
Type:	const ara::com::ServiceVersionType
Syntax:	static const ara::com::ServiceVersionType serviceVersion;
Description:	Service Version

⌋

8.2 Header: {<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h

[SWS_CM_11503] Service Proxy Header File: file name, includes and multiple inclusion guard

Upstream requirements: RS_CM_00001, RS_AP_00114, RS_AP_00116

⌈

Kind:	Header File	
Syntax:	{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h	
Description:	The Service Proxy Header File is the required (client side) API interface of a ServiceInterface. It contains different classes representing the ServiceInterface itself and its elements event, method, field and trigger. For each modeled ServiceInterface a Service Proxy Header File is generated according to this directory path/file name convention and shall: 1. Insert a multiple inclusion guard around the whole header file as per [SWS_CORE_90002] 2. Include the Service Common Header File as per [SWS_CM_01012]	
Descriptors:	{<si-namespace-derived-directory-path-lower>}	as per {<si-namespace-derived-directory-path-lower>} in [SWS_CM_01012]
	{<si-shortname-lower>}	as per {<si-shortname-lower>} in [SWS_CM_01012]

Example:	`// File=n/n_plus_1/n_plus_2/si_proxy.h (1)`
	`#ifndef N_NPLUS1_NPLUS2_SI_PROXY_H_ (2)`
	`#define N_NPLUS1_NPLUS2_SI_PROXY_H_ (2)`
	`#include ".../path/to/si_common.h" (3)`
	`...`
	`#endif // N_NPLUS1_NPLUS2_SI_PROXY_H_ (2)`	

⌋

8.2.1 Namespaces

8.2.1.1 {<hierarchical-namespace-list-lower-proxy>}

[SWS_CM_11504] Service Proxy Header File: service namespace

Upstream requirements: RS_CM_00002

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"

▽

△

Scope:	--
Syntax:	`namespace {<hierarchical-namespace-list-lower-proxy>}`
Description:	The generator shall use the `SymbolProps` aggregated in the role `PortInterface`. `namespace` to construct the encapsulating C++ namespace hierarchy as per [SWS_CM_11500].

⌋

8.2.1.2 {<hierarchical-namespace-list-lower-proxy>}::proxy

[SWS_CM_01007] `Service Proxy Header File`: proxy namespace

Upstream requirements: RS_CM_00002

⌈

Kind:	namespace
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"
Scope:	`namespace {<hierarchical-namespace-list-lower-proxy>}`
Syntax:	`namespace proxy`
Description:	Inner namespace for definitions on proxy level

⌋

8.2.1.3 {<hierarchical-namespace-list-lower-proxy>}::proxy::events

[SWS_CM_98447] `Service Proxy Header File`: events namespace

Upstream requirements: RS_CM_00002

⌈

Kind:	namespace
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"
Scope:	`namespace {<hierarchical-namespace-list-lower-proxy>}::proxy`
Syntax:	`namespace events`
Description:	Inner namespace for definitions on events level

⌋

8.2.1.4 {<hierarchical-namespace-list-lower-proxy>}::proxy::fields

[SWS_CM_98444] Service Proxy Header File: fields namespace

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	namespace {<hierarchical-namespace-list-lower-proxy>}::proxy
Syntax:	namespace fields
Description:	Inner namespace for definitions on fields level.

⌋

8.2.1.5 {<hierarchical-namespace-list-lower-proxy>}::proxy::methods

[SWS_CM_01015] Service Proxy Header File: methods namespace

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	namespace {<hierarchical-namespace-list-lower-proxy>}::proxy
Syntax:	namespace methods
Description:	Inner namespace for definitions on methods level

⌋

8.2.1.6 {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers

[SWS_CM_11505] `Service Proxy Header File`: triggers namespace

Upstream requirements: RS_CM_00002

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	namespace {<hierarchical-namespace-list-lower-proxy>}::proxy
Syntax:	namespace triggers
Description:	Inner namespace for definitions on triggers level

⌋

8.2.2 Class: {<event-name-upper-camel>}

[SWS_CM_00005] Service Proxy event class

Upstream requirements: RS_CM_00103

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy_fwd.h"	
Scope:	namespace {<hierarchical-namespace-list-lower-proxy>}::proxy::events	
Symbol:	{<event-name-upper-camel>}	
Syntax:	class {<event-name-upper-camel>} final {...};	
Description:	Service Proxy Event class, one for each VariableDataPrototype defined in the ServiceInterface in the role {	
Descriptors:	{<event-name-upper-camel>}	Name of an event created from the VariableDataPrototype. shortName defined in the role ServiceInterface. event converted to upper camel-case letters.

⌋

8.2.2.1 Public Member Types

8.2.2.1.1 Type Alias: SampleType

[SWS_CM_11401] Definition of API type {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SampleType

Upstream requirements: RS_CM_00103

⌈

Kind:	type alias	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Symbol:	SampleType	
Syntax:	using SampleType = {<p-sample-cppidt-symbol>};	
Description:	As per [SWS_CM_11400]	
Descriptors:	{<p-sample-cppidt-symbol>}	As per {<p-sample-cppidt-symbol>} in [SWS_CM_11400]

⌋

8.2.2.2 Public Member Functions

8.2.2.2.1 Member Functions

8.2.2.2.1.1 SetTransportFaultConditionHandler

[SWS_CM_00100] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetTransportFaultConditionHandler

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Syntax:	template <typename ExecutorT> void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler, ExecutorT &&executor) noexcept;	
Parameters (in):	handler	Handler of type TransportFaultConditionHandler to be set

▽

△

	executor	Executor object in which any asynchronous computation spawn shall be invoked
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Execution Context overload for SetTransportFaultConditionHandler for a given Proxy Event.	
See also:	[SWS_CM_00099]	

⌋

8.2.2.2.1.2 SetTransportFaultConditionHandler

[SWS_CM_00099] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetTransportFaultConditionHandler`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h`"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}`	
Syntax:	`void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler) noexcept;`	
Parameters (in):	handler	Handler of type TransportFaultConditionHandler to be set
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Provides possibility to register a TransportFaultConditionHandler for a given Proxy Event.	

⌋

8.2.2.2.1.3 UnsetTransportFaultConditionHandler

[SWS_CM_00105] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::UnsetTransportFaultConditionHandler`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}
Syntax:	void UnsetTransportFaultConditionHandler () noexcept;
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Provides the possibility to unregister a previously set TransportFaultConditionHandler for a given Proxy Event.

⌋

8.2.2.2.2 E2E Protection

8.2.2.2.2.1 GetE2EStateMachineState

[SWS_CM_11554] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::GetE2EStateMachineState`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Syntax:	ara::com::e2e::SMState GetE2EStateMachineState () const noexcept;	
Return value:	ara::com::e2e::SMState	State of the specific event class
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns the state of the specific event class, which was determined by the last run of E2E_Check() function invoked during the last call of GetNewSamples()	

▽

△

See also:	[SWS_CM_90417]

⌋

8.2.2.2.3 Service Communication

8.2.2.2.3.1 GetFreeSampleCount

[SWS_CM_00705] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::GetFreeSampleCount

Upstream requirements: RS_CM_00202, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00132, RS_AP_00139

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Syntax:	std::size_t GetFreeSampleCount () const noexcept;	
Return value:	std::size_t	The number of free/unused slots for event sample data in the local cache
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Query Free Sample Slots	

⌋

8.2.2.2.3.2 GetNewSamples

[SWS_CM_00701] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::GetNewSamples`

Upstream requirements: RS_CM_00202, RS_AP_00114, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00139

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Syntax:	template <typename F> ara::core::Result< std::size_t > GetNewSamples (F &&f, std::size_t maxNumberOfSamples=std::numeric_limits< std::size_t >::max()) noexcept;	
Template param:	F	Functor which shall process the received samples
Parameters (in):	f	Callable ([[func.wrap]]) f with signature void(ara::com::SamplePtr<SampleType const>)
	maxNumberOfSamples	Maximum number of received data samples being processed in the function call.
Return value:	ara::core::Result< std::size_t >	• If successful: an ara::core::Result containing a ara::core::Result::value_type i.e. a std::size_t the total number of data samples passed to f • If unsuccessful: an ara::core::Result containing an ara::core::Result::error_type i.e. a corresponding ara::com::ComErrc
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kMaxSamplesExceeded	no_rollback_semantics
		Application holds more ara::com::SamplePtrs than commited in Subscribe() Samples were possibly skipped.
Description:	Method to update the event cache.	

⌋

8.2.2.2.4 Service Discovery

8.2.2.2.4.1 GetSubscriptionState

[SWS_CM_00316] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::GetSubscriptionState`

Upstream requirements: RS_CM_00106

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Syntax:	ara::com::SubscriptionState GetSubscriptionState () const noexcept;	
Return value:	ara::com::SubscriptionState	The state of the subscription.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Query ara::com::SubscriptionState, as per [SWS_CM_00310].	

⌋

8.2.2.2.4.2 SetSubscriptionStateChangeHandler

[SWS_CM_11354] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetSubscriptionStateChangeHandler`

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Syntax:	template <typename ExecutorT> void SetSubscriptionStateChangeHandler (ara::com::SubscriptionStateChangeHandler handler, ExecutorT &&executor) noexcept;	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_00333]
	executor	As per [SWS_CM_11360]

▽

△

Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_00333] but the method shall execute in a provided context	

⌋

8.2.2.2.4.3 SetSubscriptionStateChangeHandler

[SWS_CM_00333] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetSubscriptionStateChangeHandler`

Upstream requirements: RS_CM_00106

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}`	
Syntax:	`void SetSubscriptionStateChangeHandler (ara::com::SubscriptionStateChangeHandler handler) noexcept;`	
Parameters (in):	handler	The handler that shall be called by Communication Management when the `ara::com::SubscriptionState` of the event has changed.
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Set `ara::com::SubscriptionState` change handler	

⌋

8.2.2.2.4.4 Subscribe

[SWS_CM_00141] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::Subscribe

Upstream requirements: RS_CM_00103

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Syntax:	ara::core::Result< void > Subscribe (std::size_t maxSampleCount) noexcept;	
Parameters (in):	maxSampleCount	The cacheSize of the subscription.
Return value:	ara::core::Result< void >	• If successful: an ara::core::Result containing a ara::core::Result::value_type containing a void • If unsuccessful: an ara::core::Result containing an ara::core::Result::error_type i.e. a corresponding ara::com::ComErrc
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kMaxSampleCountNotRealizable	rollback_semantics
		Provided maxSampleCount for event re-subscription does not match the maxSampleCount for the current subscription
Violations:	InsufficientPermissionsViolation	Refused by Grant Enforcement
Description:	Method to subscribe to a service event. If the Event is already subscribed to at the time of the call, and the provided maxSampleCount value is the same as for the current subscription, the method shall return without action.	

⌋

8.2.2.2.4.5 UnsetSubscriptionStateChangeHandler

[SWS_CM_00334] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::UnsetSubscriptionStateChangeHandler

Upstream requirements: RS_CM_00106, RS_AP_00114, RS_AP_00120

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"

▽

△

Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}`
Syntax:	`void UnsetSubscriptionStateChangeHandler () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Unset `ara::com::SubscriptionState` change handler

⌋

8.2.2.2.4.6 Unsubscribe

[SWS_CM_00151] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::Unsubscribe`

Upstream requirements: RS_CM_00104

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}`	
Syntax:	`void Unsubscribe () noexcept;`	
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`DanglingSamplesViolation`	Precondition [SWS_CM_00085] has been violated.
Description:	Method to unsubscribe from a service event. If the Event is not subscribed to at the time of the call, `Unsubscribe()` shall return without action.	

⌋

8.2.2.2.5 Service Management

8.2.2.2.5.1 SetReceiveHandler

[SWS_CM_00181] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetReceiveHandler`

Upstream requirements: RS_CM_00203, RS_AP_00114, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}	
Syntax:	void SetReceiveHandler (ara::com::EventReceiveHandler handler) noexcept;	
Parameters (in):	handler	The handler that shall be called by Communication Management upon event arrival. The EventReceiveHandler constitutes a function without parameters and has to use the f method of the specific Event class to access the retrieved event data. See [SWS_CM_00309] for its definition.
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InvalidHandlerViolation	Provided Callable handler does not exist or is null
Description:	Set/Register the event receive handler	
See also:	[SWS_CM_00250], [SWS_CM_11356]	

⌋

8.2.2.2.5.2 SetReceiveHandler

[SWS_CM_11356] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetReceiveHandler`

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"

▽

△

Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}`	
Syntax:	`template <typename ExecutorT> void SetReceiveHandler (ara::com::EventReceiveHandler handler, ExecutorT &&executor) noexcept;`	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_00181]
	executor	As per [SWS_CM_11360]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_00181] but the method shall execute in a provided context	
See also:	[SWS_CM_00181], [SWS_CM_00250]	

⌋

8.2.2.2.5.3 UnsetReceiveHandler

[SWS_CM_00183] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}`::proxy::events::`{<event-name-upper-camel>}`::UnsetReceiveHandler

Upstream requirements: RS_CM_00203

⌈

Kind:	function
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}`
Syntax:	`void UnsetReceiveHandler () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Disable service event trigger

⌋

8.2.3 Class: {<field-name-upper-camel>}

[SWS_CM_00008] Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}

Upstream requirements: RS_CM_00216, RS_AP_00114

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy_fwd.h"	
Scope:	namespace {<hierarchical-namespace-list-lower-proxy>}::proxy::fields	
Symbol:	{<field-name-upper-camel>}	
Syntax:	class {<field-name-upper-camel>} final {...};	
Description:	Service proxy Field class, one for each Field defined in the ServiceInterface in the role field.	
Descriptors:	{<field-name-upper-camel>}	As per {<field-name-upper-camel>} in [SWS_CM_00007]

⌋

8.2.3.1 Public Member Types

8.2.3.1.1 Type Alias: FieldType

[SWS_CM_11403] Definition of API type {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::FieldType

Upstream requirements: RS_CM_00216, RS_AP_00114

⌈

Kind:	type alias	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Symbol:	FieldType	
Syntax:	using FieldType = {<p-field-cppidt-symbol>};	
Description:	As per [SWS_CM_11402]	
Descriptors:	{<p-field-cppidt-symbol>}	As per {<p-field-cppidt-symbol>} in [SWS_CM_11402]

⌋

8.2.3.2 Public Member Functions

8.2.3.2.1 Member Functions

8.2.3.2.1.1 GetE2EStateMachineState

[SWS_CM_11612] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::GetE2EStateMachineState`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	ara::com::e2e::SMState GetE2EStateMachineState () const noexcept;	
Return value:	ara::com::e2e::SMState	State of the specific field class
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns the state of the specific field class, which was determined by the last run of E2E_Check() function invoked during the last call of GetNewSamples()	
See also:	[SWS_CM_90417]	

⌋

8.2.3.2.1.2 SetReceiveHandler

[SWS_CM_11611] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetReceiveHandler`

Upstream requirements: RS_CM_00211, RS_AP_00151, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}

▽

△

Syntax:	`template <typename ExecutorT>` `void SetReceiveHandler (ara::com::FieldReceiveHandler handler, ExecutorT &&executor) noexcept;`	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_11605]
	executor	As per [SWS_CM_11360]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_11605] but the handler shall execute in a provided context	
See also:	[SWS_CM_11605], [SWS_CM_00250], [SWS_CM_11356]	

⌋

8.2.3.2.1.3 SetTransportFaultConditionHandler

[SWS_CM_00109] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}`::proxy::fields::`{<field-name-upper-camel>}`::SetTransportFaultConditionHandler

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}`	
Syntax:	`void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler) noexcept;`	
Parameters (in):	handler	Handler of type TransportFaultConditionHandler to be set
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Provides possibility to register a TransportFaultConditionHandler for a given Proxy Field Notifier.	

⌋

8.2.3.2.1.4 SetTransportFaultConditionHandler

[SWS_CM_00110] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetTransportFaultConditionHandler

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	template <typename ExecutorT> void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler, ExecutorT &&executor) noexcept;	
Parameters (in):	handler	Handler to be set
	executor	Executor object in which any asynchronous computation spawn shall be invoked
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InvalidHandlerViolation	Provided Callable handler does not exist or is null
Description:	Execution Context overload for SetTransportFaultConditionHandler for a given Proxy Field Notifier.	
See also:	[SWS_CM_00109]	

⌋

8.2.3.2.1.5 UnsetTransportFaultConditionHandler

[SWS_CM_00126] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::UnsetTransportFaultConditionHandler

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}
Syntax:	void UnsetTransportFaultConditionHandler () noexcept;
Return value:	None

▽

△

Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Provides the possibility to unregister a previously set TransportFaultConditionHandler for a given Proxy Field Notifier.

⌋

8.2.3.2.2 Service Communication

8.2.3.2.2.1 Get

[SWS_CM_00112] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::Get`

Upstream requirements: RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	ara::core::Future< FieldType > Get () noexcept;	
Return value:	ara::core::Future< Field Type >	• If successful: an ara::core::Future containing a FieldType object as per [SWS_CM_11403] • If unsuccessful: an ara::core::Future containing a corresponding ara::com::future_errc
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kNetwork BindingFailure	rollback_semantics The network binding reported a recoverable communications error or a secure communication failure
Violations:	InsufficientPermissionsViolation	Refused by Grant Enforcement
Description:	Get the current value of the Field. The method is present only if Field.hasGetter ==TRUE, otherwise it is not present (shall not be generated)	

⌋

8.2.3.2.2.2 GetFreeSampleCount

[SWS_CM_11603] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::GetFreeSampleCount`

Upstream requirements: RS_CM_00202, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00132, RS_AP_00139

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	std::size_t GetFreeSampleCount () const noexcept;	
Return value:	std::size_t	The number of free/unused slots for field notification sample data in the local cache
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Query Free Sample Slots	

⌋

8.2.3.2.2.3 GetNewSamples

[SWS_CM_11610] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::GetNewSamples`

Upstream requirements: RS_CM_00202, RS_AP_00114, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00139

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	template <typename F> ara::core::Result< std::size_t > GetNewSamples (F &&f, std::size_t maxNumberOfSamples=std::numeric_limits< std::size_t >::max());	
Template param:	F	As per [SWS_CM_00701]
Parameters (in):	f	As per [SWS_CM_00701]
	maxNumberOfSamples	As per [SWS_CM_00701]
Return value:	ara::core::Result< std::size_t >	As per [SWS_CM_00701]

▽

△

Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kMaxSamples Exceeded	rollback_semantics
		As per [SWS_CM_00701]
Description:	Method to update the field notification cache.	

⌋

8.2.3.2.2.4 Set

[SWS_CM_00113] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::Set`

Upstream requirements: RS_CM_00217, RS_AP_00114, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h`"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}`	
Syntax:	`ara::core::Future< FieldType > Set ({<p-field-derived-type>} value) noexcept;`	
Parameters (in):	value	New value of field
Return value:	ara::core::Future< FieldType >	• If successful: an `ara::core::Future` containing a `FieldType` object as per [SWS_CM_11403] • If unsuccessful: an `ara::core::Future` containing a corresponding `ara::com::future_errc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kNetwork BindingFailure	no_rollback_semantics
		The network binding reported a recoverable communications error or a secure communication failure. In case of a communication timeout this could lead to an uncertain state on the remote server side.
Violations:	`InsufficientPermissionsViolation`	Refused by Grant Enforcement
Description:	Set a new value of the `Field`. The method is present only if `Field.hasSetter==TRUE`, otherwise it is not present (shall not be generated). There is no need to have Get method itself return [Application Errors] errors when getting a field value (as there are no errors possible when getting a field value). The Set-Method may return [Application Errors], but does so via returning a value different from the one passed in the request parameter.	
Descriptors:	`{<p-field-derived-type>}`	As per `{<derived-type>}` in [SWS_CM_00191]

⌋

8.2.3.2.3 Service Discovery

8.2.3.2.3.1 GetSubscriptionState

[SWS_CM_11604] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::GetSubscriptionState`

Upstream requirements: RS_CM_00106

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	ara::com::SubscriptionState GetSubscriptionState () const noexcept;	
Return value:	ara::com::SubscriptionState	The state of the subscription.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Query ara::com::SubscriptionState, as per [SWS_CM_00310].	

⌋

8.2.3.2.3.2 SetSubscriptionStateChangeHandler

[SWS_CM_11608] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetSubscriptionStateChangeHandler`

Upstream requirements: RS_CM_00211, RS_AP_00151, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	template <typename ExecutorT> void SetSubscriptionStateChangeHandler (ara::com::SubscriptionStateChangeHandler handler, ExecutorT &&executor) noexcept;	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_11354]
	executor	As per [SWS_CM_11360]

▽

△

Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_11607] but the handler shall execute in a provided context	

⌋

8.2.3.2.3.3 SetSubscriptionStateChangeHandler

[SWS_CM_11607] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetSubscriptionStateChangeHandler`

Upstream requirements: RS_CM_00106

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}`	
Syntax:	`void SetSubscriptionStateChangeHandler (ara::com::SubscriptionStateChangeHandler handler) noexcept;`	
Parameters (in):	handler	As per [SWS_CM_00333]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Set `ara::com::SubscriptionState` change handler	

⌋

8.2.3.2.3.4 Subscribe

[SWS_CM_11601] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::Subscribe

Upstream requirements: RS_CM_00103

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	ara::core::Result< void > Subscribe (std::size_t maxSampleCount) noexcept;	
Parameters (in):	maxSampleCount	As per [SWS_CM_00141]
Return value:	ara::core::Result< void >	As per [SWS_CM_00141]
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kMaxSampleCountNotRealizable	rollback_semantics
		As per [SWS_CM_00141]
Violations:	InsufficientPermissionsViolation	Refused by Grant Enforcement
Description:	Method to subscribe to a service field notification. If the Field is already subscribed to at the time of the call, and the provided maxSampleCount value is the same as for the current subscription, the method shall return without action.	

⌋

8.2.3.2.3.5 UnsetSubscriptionStateChangeHandler

[SWS_CM_11609] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::UnsetSubscriptionStateChangeHandler

Upstream requirements: RS_CM_00106, RS_AP_00114, RS_AP_00120

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}
Syntax:	void UnsetSubscriptionStateChangeHandler () noexcept;
Return value:	None

▽

△

Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Unset `ara::com::SubscriptionState` change handler

⌋

8.2.3.2.3.6 Unsubscribe

[SWS_CM_11602] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::Unsubscribe`

Upstream requirements: RS_CM_00104

⌈

Kind:	function
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h`"
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}`
Syntax:	`void Unsubscribe () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Method to unsubscribe from a service field notification. If the Field is not subscribed to at the time of the call, `Unsubscribe()` shall return without action.

⌋

8.2.3.2.4 Service Management

8.2.3.2.4.1 SetReceiveHandler

[SWS_CM_11605] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetReceiveHandler

Upstream requirements: RS_CM_00203, RS_AP_00114, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}	
Syntax:	void SetReceiveHandler (ara::com::FieldReceiveHandler handler) noexcept;	
Parameters (in):	handler	The handler that shall be called by Communication Management upon notification arrival. The FieldReceiveHandler constitutes a function without parameters and has to use the f method of the specific Field class to access the retrieved event data. See [SWS_CM_11615] for its definition.
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InvalidHandlerViolation	Provided Callable handler does not exist or is null
Description:	Set/Register the Field notification receive handler	
See also:	[SWS_CM_00181], [SWS_CM_00250], [SWS_CM_11356]	

⌋

8.2.3.2.4.2 UnsetReceiveHandler

[SWS_CM_11606] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::UnsetReceiveHandler

Upstream requirements: RS_CM_00203

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}

▽

△

Syntax:	`void UnsetReceiveHandler () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Disable service field notification trigger

⌋

8.2.4 Class: {<fnfmethod-name-upper-camel>}

[SWS_CM_99332] Definition of API class `{<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<fnfmethod-name-upper-camel>}`

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_AP_00114

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy_fwd.h"	
Scope:	`namespace {<hierarchical-namespace-list-lower-proxy>}::proxy::methods`	
Symbol:	`{<fnfmethod-name-upper-camel>}`	
Syntax:	`class {<fnfmethod-name-upper-camel>} {...};`	
Description:	Service proxy fire-and-forget method class, one for each `ClientServerOperation` (`ClientServerOperation.fireAndForget==TRUE`) defined in the `ServiceInterface` in the role `method`.	
Descriptors:	`{<fnfmethod-name-upper-camel>}`	Name of a method (`ClientServerOperation.fireAndForget==TRUE`) created from the `ClientServerOperation`. `shortName` defined in the `ServiceInterface` in the role `method` converted to upper camel-case letters.

⌋

8.2.4.1 Public Member Functions

8.2.4.1.1 Service Communication

8.2.4.1.1.1 operator()

[SWS_CM_90435] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<fnfmethod-name-upper-camel>}::operator()

Upstream requirements: RS_CM_00225, RS_AP_00114, RS_AP_00120

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<fnfmethod-name-upper-camel>}	
Syntax:	void operator() ({<p-fnfmethod-in-arg-derived-type-0toN>} {<fnfmethod-in-arg-symbol-0toN>}) noexcept;	
Parameters (in):	{<fnfmethod-in-arg-symbol-0toN>}	As per {<p-method-in-arg-symbol-0toN>} in [SWS_CM_00191]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Initiate a fire-and-forget method call (ClientServerOperation. fireAndForget ==TRUE)	
Descriptors:	{<p-fnfmethod-in-arg-derived-type-0toN>}	As per {<p-method-in-arg-derived-type-0toN>} in [SWS_CM_00191]

⌋

8.2.5 Class: {<method-name-upper-camel>}

[SWS_CM_00006] Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<method-name-upper-camel>}

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_AP_00114

⌈

Kind:	class
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy_fwd.h"

▽

△

Scope:	namespace {<hierarchical-namespace-list-lower-proxy>}::proxy::methods	
Symbol:	{<method-name-upper-camel>}	
Syntax:	class {<method-name-upper-camel>} final {...};	
Description:	Service proxy method class, one for each ClientServerOperation (ClientServer Operation.fireAndForget ==FALSE) defined in the ServiceInterface in the role method.	
Descriptors:	{<method-name-upper-camel>}	As per {<method-name-upper-camel>} in [SWS_CM_00191]

⌋

8.2.5.1 Public Member Functions

8.2.5.1.1 Service Communication

8.2.5.1.1.1 operator()

[SWS_CM_00196] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<method-name-upper-camel>}::operator()

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<method-name-upper-camel>}	
Syntax:	ara::core::Future< Output > operator() ({<p-method-in-arg-derived-type-0toN>} {<p-method-in-arg-symbol-0toN>}) noexcept;	
Parameters (in):	{<p-method-in-arg-symbol-0toN>}	As per {<p-method-in-arg-symbol-0toN>} in [SWS_CM_00191]
Return value:	ara::core::Future< Output >	• If successful: an ara::core::Future containing a Output object as per [SWS_CM_99333] • If unsuccessful: an ara::core::Future containing a corresponding ara::com::future_errc
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kNetwork BindingFailure	no_rollback_semantics The network binding reported a recoverable communications error or a secure communication failure. In case of a communication timeout this could lead to an uncertain state on the remote server side.

▽

△

	ComErrc::kUnknown ApplicationError	rollback_semantics
		A remote service returned an unconfigured application error.
	ComErrc::kServiceNot Available	rollback_semantics
		Service is not available after being previously offered via `OfferService()`
	ComE2EErrc::k UnspecifiedE2EError	rollback_semantics
		A remote service returned an unspecified E2E error.
Violations:	`InsufficientPer-missionsViolation`	Refused by Grant Enforcement
Description:	Initiate a method call (`ClientServerOperation`. `fireAndForget` ==FALSE). The method call will return immediately. The caller's selection of a synchronous or asynchronous behavior to get the method output is achieved by the use of the returned `ara::core::Future` object which is used to query for method completion and result including possible error.	
Descriptors:	`{<p-method-in-arg-derived-type-0toN>}`	As per `{<p-method-in-arg-derived-type-0toN>}` in [SWS_CM_00191]

⌋

8.2.6 Struct: Output

[SWS_CM_99333] Definition of API class `{<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<method-name-upper-camel>}::Output`

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	struct	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Forwarding header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy_fwd.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<method-name-upper-camel>}`	
Symbol:	Output	
Syntax:	`struct Output {...};`	
Description:	Structure wrapping the *out* arguments for a `method`	
Descriptors:	`{<p-method-out-args-members-ordered>}`	As per `{<p-method-out-args-members-ordered>}` in [SWS_CM_99556]

⌋

8.2.7 Class: {<trigger-name-upper-camel>}

[SWS_CM_00727] Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}

Upstream requirements: RS_CM_00103, RS_AP_00114

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy_fwd.h"	
Scope:	namespace {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers	
Symbol:	{<trigger-name-upper-camel>}	
Syntax:	class {<trigger-name-upper-camel>} final {...};	
Description:	Service proxy Trigger class, one for each Trigger defined in the ServiceInterface in the role trigger.	
Descriptors:	{<trigger-name-upper-camel>}	Name of a trigger created from the Trigger.shortName defined in the ServiceInterface in the role trigger converted to upper camel-case letters.

⌋

8.2.7.1 Public Member Functions

8.2.7.1.1 Member Functions

8.2.7.1.1.1 SetTransportFaultConditionHandler

[SWS_CM_00107] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::SetTransportFaultConditionHandler

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}	
Syntax:	template <typename ExecutorT> void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler, ExecutorT &&executor) noexcept;	
Parameters (in):	handler	Handler to be set

▽

△

	executor	Executor object in which any asynchronous computation spawn shall be invoked
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Execution Context overload for SetTransportFaultConditionHandler for a given Proxy Trigger.	
See also:	[SWS_CM_00106]	

⌋

8.2.7.1.1.2 SetTransportFaultConditionHandler

[SWS_CM_00106] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::SetTransportFaultConditionHandler`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h`"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}`	
Syntax:	`void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler) noexcept;`	
Parameters (in):	handler	Handler of type TransportFaultConditionHandler to be set
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Provides possibility to register a TransportFaultConditionHandler for a given Proxy Trigger.	

⌋

8.2.7.1.1.3 UnsetTransportFaultConditionHandler

[SWS_CM_00108] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::UnsetTransportFaultConditionHandler

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}
Syntax:	void UnsetTransportFaultConditionHandler () noexcept;
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Provides the possibility to unregister a previously set TransportFaultConditionHandler for a given Proxy Trigger.

⌋

8.2.7.1.2 Service Communication

8.2.7.1.2.1 GetNewTriggers

[SWS_CM_00226] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::GetNewTriggers

Upstream requirements: RS_CM_00202, RS_AP_00114, RS_AP_00120, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00139

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}	
Syntax:	std::size_t GetNewTriggers () noexcept;	
Return value:	std::size_t	The number of triggers occurred since the last call to GetNewTriggers (a Zero value means that no new triggers have been received)
Exception Safety:	exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Method to update the trigger counter.

⌋

8.2.7.1.3 Service Discovery

8.2.7.1.3.1 Subscribe

[SWS_CM_00723] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::Subscribe`

Upstream requirements: RS_CM_00103, RS_AP_00114, RS_AP_00120

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}	
Syntax:	void Subscribe () noexcept;	
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InsufficientPermissionsViolation	Refused by Grant Enforcement
Description:	Method to subscribe to a service trigger. When called it starts a subscription of the corresponding Trigger. If the Trigger is already subscribed to at the time of the call, and this method is invoked, it shall return without any action.	

⌋

8.2.7.1.3.2 Unsubscribe

[SWS_CM_00810] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::Unsubscribe

Upstream requirements: RS_CM_00103, RS_AP_00114, RS_AP_00120

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}
Syntax:	void Unsubscribe () noexcept;
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Method to unsubscribe from a service trigger. If the Trigger is not subscribed to at the time of the call invoked, it shall return without any action.

⌋

8.2.7.1.4 Service Management

8.2.7.1.4.1 SetReceiveHandler

[SWS_CM_00352] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::SetReceiveHandler

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}	
Syntax:	template <typename ExecutorT> void SetReceiveHandler (ara::com::TriggerReceiveHandler handler, ExecutorT &&executor) noexcept;	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_00181]
	executor	As per [SWS_CM_11360]

▽

△

Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_00250] but the method shall execute in a provided context	
See also:	[SWS_CM_11356], [SWS_CM_00250]	

⌋

8.2.7.1.4.2 SetReceiveHandler

[SWS_CM_00250] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::SetReceiveHandler`

Upstream requirements: RS_CM_00203, RS_AP_00114, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}`	
Syntax:	`void SetReceiveHandler (ara::com::TriggerReceiveHandler handler) noexcept;`	
Parameters (in):	handler	The handler that shall be called by Communication Management upon trigger arrival. The `ara::com::TriggerReceiveHandler` constitutes a function without parameters and has to use the `GetNewTriggers` method of the specific Trigger class to access the retrieved trigger counter. See [SWS_CM_00351] for it's definition.
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Trigger SetReceiveHandler	
See also:	[SWS_CM_00181], [SWS_CM_11356]	

⌋

8.2.7.1.4.3 UnsetReceiveHandler

[SWS_CM_11614] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::UnsetReceiveHandler

Upstream requirements: RS_CM_00203

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}
Syntax:	void UnsetReceiveHandler ();
Return value:	None
Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	Disable Trigger receive handler
See also:	[SWS_CM_00183]

⌋

8.2.8 Class: {<service-interface-name-upper-camel>}Proxy

[SWS_CM_00004] Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00122

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy_fwd.h"	
Scope:	namespace {<hierarchical-namespace-list-lower-proxy>}::proxy	
Symbol:	{<service-interface-name-upper-camel>}Proxy	
Syntax:	class {<service-interface-name-upper-camel>}Proxy final {...};	
Description:	Service Interface Proxy	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.1 Public Member Variables

8.2.8.1.1 {<event-name-upper-camel>}

[SWS_CM_99445] Definition of API variable {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::{<event-name-upper-camel>}

Upstream requirements: RS_CM_00201, RS_AP_00114

⌈

Kind:	variable	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Symbol:	{<event-name-upper-camel>}	
Type:	events::{<event-name-upper-camel>}	
Syntax:	events::{<event-name-upper-camel>} {<event-name-upper-camel>};	
Description:	Event member, one for each VariableDataPrototype defined in the ServiceInterface in the role event.	
Descriptors:	{<event-name-upper-camel>}	As per {<event-name-upper-camel>} in [SWS_CM_00005]

⌋

8.2.8.1.2 {<field-name-upper-camel>}

[SWS_CM_99446] Definition of API variable {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::{<field-name-upper-camel>}

Upstream requirements: RS_CM_00216, RS_AP_00114

⌈

Kind:	variable
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy
Symbol:	{<field-name-upper-camel>}
Type:	fields::{<field-name-upper-camel>}
Syntax:	fields::{<field-name-upper-camel>} {<field-name-upper-camel>};
Description:	Field member, one for each Field defined in the ServiceInterface in the role field.

▽

△

Descriptors:	{<field-name-upper-camel>}	As per {<field-name-upper-camel>} in [SWS_CM_00007]

⌋

8.2.8.1.3 {<method-name-upper-camel>}

[SWS_CM_99447] Definition of API variable {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::{<method-name-upper-camel>}

Upstream requirements: RS_CM_00212, RS_CM_00213, RS_AP_00114

⌈

Kind:	variable	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Symbol:	{<method-name-upper-camel>}	
Type:	methods::{<method-name-upper-camel>}	
Syntax:	methods::{<method-name-upper-camel>} {<method-name-upper-camel>};	
Description:	Method member, one for each ClientServerOperation defined in the ServiceInterface in the role method.	
Descriptors:	{<method-name-upper-camel>}	As per {<method-name-upper-camel>} in [SWS_CM_00191]

⌋

8.2.8.2 Public Member Functions

8.2.8.2.1 Special Member Functions

8.2.8.2.1.1 Copy Assignment Operator

[SWS_CM_11551] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::operator=

Upstream requirements: RS_CM_00102, RS_AP_00114

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	{<service-interface-name-upper-camel>}Proxy & operator= (const {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy &other)=delete;	
Description:	Copy assignment constructor deletion	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.1.2 Move Assignment Operator

[SWS_CM_11552] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::operator=

Upstream requirements: RS_CM_00102, RS_AP_00114

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy
Syntax:	{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy & operator= ({<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy &&other) noexcept;

▽

△

Parameters (in):	other	Other object to move
Return value:	{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy &	The moved object
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move assignment constructor	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.1.3 Copy Constructor

[SWS_CM_00136] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::{<service-interface-name-upper-camel>}Proxy

Upstream requirements: RS_CM_00102, RS_AP_00114

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	{<service-interface-name-upper-camel>}Proxy (const {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy &other)=delete;	
Description:	Copy Constructor deletion	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.1.4 Move Constructor

[SWS_CM_00137] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::{<service-interface-name-upper-camel>}Proxy

Upstream requirements: RS_CM_00102, RS_AP_00114

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	{<service-interface-name-upper-camel>}Proxy ({<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy &&other);	
Parameters (in):	other	Other object to move
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.1.5 Destructor

[SWS_CM_99444] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::~{<service-interface-name-upper-camel>}Proxy

Upstream requirements: RS_CM_00102, RS_AP_00114

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	~{<service-interface-name-upper-camel>}Proxy ();	
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Violations:	DanglingSamplesViolation	Precondition [SWS_CM_00087] has been violated.

▽

△

Description:	Destruction of a `Proxy`	
Descriptors:	`{<service-interface-name-upper-camel>}`	As per `{<service-interface-name-upper-camel>}` in [SWS_CM_00002]

⌋

8.2.8.2.2 Constructors

8.2.8.2.2.1 {<service-interface-name-upper-camel>}Proxy

[SWS_CM_00131] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}`::proxy::`{<service-interface-name-upper-camel>}`Proxy::`{<service-interface-name-upper-camel>}`Proxy

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h`"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy`	
Syntax:	`explicit {<service-interface-name-upper-camel>}Proxy (const HandleType &handle);`	
Parameters (in):	handle	A found `FindServiceHandle` returned by **any** of the `FindService()` methods: • [SWS_CM_00122] • [SWS_CM_00622] or a `FindServiceHandle` found and sent to the assigned `FindServiceHandler` callback, by **any** of the `StartFindService()` methods: • [SWS_CM_00123] • [SWS_CM_00623] • [SWS_CM_11352] • [SWS_CM_11365]
Exceptions:	`ara::com::ComException`	As per `ara::com::ComErrc::kServiceNotAvailable`
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Violations:	`ErroneousServiceHandle`	Provided `HandleType` is `null` or corrupt
	`InsufficientPermissionsViolation`	Refused by Grant Enforcement

▽

△

Description:	Construction of the service proxy from the found handles (from **any** of the `FindService()`/ `StartFindService()` methods)	
Descriptors:	`{<service-interface-name-upper-camel>}`	As per `{<service-interface-name-upper-camel>}` in [SWS_CM_00002]

⌋

8.2.8.2.3 Named Constructors

8.2.8.2.3.1 Create

[SWS_CM_10438] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}`::proxy::`{<service-interface-name-upper-camel>}`Proxy::Create

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00121, RS_AP_00127, RS_AP_00128, RS_AP_00132, RS_AP_00139

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy`	
Syntax:	`static ara::core::Result< {<service-interface-name-upper-camel>}Proxy > Create (const {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType &handle) noexcept;`	
Parameters (in):	handle	As per [SWS_CM_00131]
Return value:	ara::core::Result< `{<service-interface-name-upper-camel>}`Proxy >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `Proxy` • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::com::ComErrc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kServiceNotAvailable	rollback_semantics Service is not available after being previously offered via `OfferService()`
	ComErrc::kNetworkBindingFailure	rollback_semantics The network binding reported a recoverable communications error or a secure communication failure
Violations:	`ErroneousServiceHandle`	Provided `HandleType` is `null` or corrupt
	`InsufficientPermissionsViolation`	Refused by Grant Enforcement

▽

△

Description:	Exception-less construction of a `Proxy` from a `HandleType` using the Named Constructor idiom.	
Descriptors:	`{<service-interface-name-upper-camel>}`	As per `{<service-interface-name-upper-camel>}` in [SWS_CM_00002]

⌋

8.2.8.2.4 Service Discovery

8.2.8.2.4.1 FindService

[SWS_CM_00122] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::FindService`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00119, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy`	
Syntax:	`static ara::core::Result< ara::com::ServiceHandleContainer< HandleType > > FindService (ara::com::InstanceIdentifier instance) noexcept;`	
Parameters (in):	instance	`ara::com::InstanceIdentifier` qualifying the wanted instance of the service
Return value:	ara::core::Result< ara::com::ServiceHandle Container< `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}`Proxy::HandleType > >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `ara::com::ServiceHandleContainer` further containing `HandleType`s for all matching services • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::com::ComErrc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kNetwork BindingFailure	rollback_semantics The network binding reported a recoverable communications error or a secure communication failure
Violations:	`InsufficientPermissionsViolation`	Refused by Grant Enforcement
Description:	Find a service using an `ara::com::InstanceIdentifier`, with immediately returned request	

▽

△

Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.4.2 FindService

[SWS_CM_00622] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::FindService

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00119, RS_AP_00121, RS_AP_00127, RS_AP_00137

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	static ara::core::Result< ara::com::ServiceHandleContainer< HandleType > > FindService (ara::core::InstanceSpecifier instance) noexcept;	
Parameters (in):	instance	ara::core::InstanceSpecifier qualifying the wanted instance of the service
Return value:	ara::core::Result< ara::com::ServiceHandleContainer< {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType > >	As per [SWS_CM_00122]
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kNetwork BindingFailure	rollback_semantics
		The network binding reported a recoverable communications error or a secure communication failure
Violations:	InstanceSpecifierMappingIntegrityViolation	InstanceSpecifier either cannot be resolved in the model in the context of the executable, or it refers to a model element other than a PortPrototype.
	PortInterfaceMappingViolation	The type of mapping does not match the expected type of Port Interface: ServiceInterface referenced by a ServiceInstanceToPortPrototypeMapping.
	ProcessMappingViolation	The type of mapping does not match the expected type of Port Interface

▽

△

	InstanceSpecifier-AlreadyInUseViola-tion	The constructor was called with an Instance Specifier already
Description:	Find a service using an `ara::core::InstanceSpecifier`, with immediately returned request	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.4.3 GetHandle

[SWS_CM_10383] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::GetHandle

Upstream requirements: RS_CM_00107, RS_AP_00114, RS_AP_00119

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	`HandleType GetHandle () const noexcept;`	
Return value:	{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType	As per [SWS_CM_00312]
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Return the handle used to create the proxy instance	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.4.4 GetServiceState

[SWS_CM_01073] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::GetServiceState`

Status: DRAFT

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	ara::core::Result< ara::com::ServiceState > GetServiceState () noexcept;	
Return value:	ara::core::Result< ara::com::ServiceState >	• If successful: an ara::core::Result containing a ara::core::Result::value_type i.e. a ara::com::ServiceState giving the service state • If unsuccessful: an ara::core::Result containing an ara::core::Result::error_type i.e. a corresponding ara::com::ComErrc
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kNetwork BindingFailure	rollback_semantics The network binding reported a recoverable communications error or a secure communication failure
Description:	Returns the state of the service	

⌋

8.2.8.2.4.5 SetServiceStateChangeHandler

[SWS_CM_01076] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::SetServiceStateChangeHandler` ⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	template <typename ExecutorT> void SetServiceStateChangeHandler (ara::com::ServiceStateHandler handler, ExecutorT &&executor) noexcept;	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_01074]

▽

△

	executor	As per [SWS_CM_11360]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerVio-lation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_01074] but the method shall execute in a provided context.	

⌋

8.2.8.2.4.6 SetServiceStateChangeHandler

[SWS_CM_01074] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}`::proxy::`{<service-interface-name-upper-camel>}`Proxy::SetServiceStateChangeHandler ⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy`	
Syntax:	`void SetServiceStateChangeHandler (ara::com::ServiceStateHandler handler) noexcept;`	
Parameters (in):	handler	The callback for service state change handler
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerVio-lation`	Provided `Callable` handler does not exist or is `null`
Description:	Register a service state change handler. This handler shall be called by the Communication Management implementation as soon as the service availability state has changed. The Handler may be overwritten during runtime.	

⌋

8.2.8.2.4.7 StartFindService

[SWS_CM_00623] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StartFindService`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00119, RS_AP_00121, RS_AP_00127, RS_AP_00137

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	static ara::core::Result< ara::com::FindServiceHandle > StartFindService (ara::com::FindServiceHandler< {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType > handler, ara::core::InstanceSpecifier instance) noexcept;	
Parameters (in):	handler	As per [SWS_CM_00123]
	instance	As per [SWS_CM_00622]
Return value:	ara::core::Result< ara::com::FindServiceHandle >	As per [SWS_CM_00123]
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kNetworkBindingFailure	rollback_semantics
		The network binding reported a recoverable communications error or a secure communication failure
Violations:	InstanceSpecifierMappingIntegrityViolation	InstanceSpecifier either cannot be resolved in the model in the context of the executable, or it refers to a model element other than a PortPrototype.
	PortInterfaceMappingViolation	The type of mapping does not match the expected type of Port Interface: ServiceInterface referenced by a ServiceInstanceToPortPrototypeMapping.
	ProcessMappingViolation	The type of mapping does not match the expected type of Port Interface
	InstanceSpecifierAlreadyInUseViolation	The constructor was called with an Instance Specifier already
	InvalidHandlerViolation	Provided Callable handler does not exist or is null
Description:	Find a service using an ara::core::InstanceSpecifier, with handler registration	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.4.8 StartFindService

[SWS_CM_11365] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StartFindService`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00119, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	template <typename ExecutorT> static ara::com::FindServiceHandle StartFindService (ara::com::FindServiceHandler< {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType > handler, ara::core::InstanceSpecifier instance, ExecutorT &&executor) noexcept;	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_00123]
	instance	As per [SWS_CM_00123]
	executor	As per [SWS_CM_11360]
Return value:	ara::com::FindServiceHandle	As per [SWS_CM_00123]
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InstanceSpecifierMappingIntegrityViolation	InstanceSpecifier either cannot be resolved in the model in the context of the executable, or it refers to a model element other than a PortPrototype.
	PortInterfaceMappingViolation	The type of mapping does not match the expected type of Port Interface: ServiceInterface referenced by a ServiceInstanceToPortPrototypeMapping.
	ProcessMappingViolation	The type of mapping does not match the expected type of Port Interface
	InstanceSpecifierAlreadyInUseViolation	The constructor was called with an Instance Specifier already
	InvalidHandlerViolation	Provided Callable handler does not exist or is null
Description:	Find a service using an ara::core::InstanceSpecifier, with handler registration, in a provided context	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.4.9 StartFindService

[SWS_CM_11352] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StartFindService`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00119, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	template <typename ExecutorT> static ara::com::FindServiceHandle StartFindService (ara::com::FindServiceHandler< {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType > handler, ara::com::InstanceIdentifier instance, ExecutorT &&executor) noexcept;	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_00123]
	instance	As per [SWS_CM_00122]
	executor	As per [SWS_CM_11360]
Return value:	ara::com::FindService Handle	As per [SWS_CM_00123]
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InvalidHandlerViolation	Provided Callable handler does not exist or is null
	InsufficientPermissionsViolation	Refused by Grant Enforcement
Description:	Find a service using an ara::com::InstanceIdentifier, with handler registration, - in a provided execution context	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.4.10 StartFindService

[SWS_CM_00123] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StartFindService`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00119, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	static ara::core::Result< ara::com::FindServiceHandle > StartFindService (ara::com::FindServiceHandler<{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType > handler, ara::com::InstanceIdentifier instance) noexcept;	
Parameters (in):	handler	A ara::com::FindServiceHandler for the corresponding ServiceProxy class, which gets called any time a AdaptivePlatformServiceInstance matching the given instance criteria is found.
	instance	As per [SWS_CM_00122]
Return value:	ara::core::Result< ara::com::FindServiceHandle >	• If successful: an ara::core::Result containing a ara::core::Result::value_type containing a ara::com::FindServiceHandle for this search/find request which is needed to stop the service availability monitoring and related firing of the given handler. • If unsuccessful: an ara::core::Result containing an ara::core::Result::error_type i.e. a corresponding ara::com::ComErrc
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kNetwork BindingFailure	rollback_semantics
		The network binding reported a recoverable communications error or a secure communication failure
Violations:	InvalidHandlerViolation	Provided Callable handler does not exist or is null
	InsufficientPermissionsViolation	Refused by Grant Enforcement
Description:	Find a service using an ara::com::InstanceIdentifier, with handler registration	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.2.8.2.4.11 StopFindService

[SWS_CM_00125] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StopFindService

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy	
Syntax:	void StopFindService (ara::com::FindServiceHandle handle) noexcept;	
Parameters (in):	handle	The ara::com::FindServiceHandle returned by StartFindService methods: • [SWS_CM_00123] • [SWS_CM_00623] • [SWS_CM_11352] • [SWS_CM_11365]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Stop an ongoing StartFindService() - Stops receiving further notifications for the provided ara::com::FindServiceHandle	

⌋

8.2.8.2.4.12 UnsetServiceStateChangeHandler

[SWS_CM_01075] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::UnsetServiceStateChangeHandler ⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy
Syntax:	void UnsetServiceStateChangeHandler () noexcept;
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	As per [SWS_CM_01074]

⌋

8.2.9 Class: HandleType

[SWS_CM_00312] Definition of API class `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00122

⌈

Kind:	class
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy_fwd.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy
Symbol:	HandleType
Syntax:	class HandleType {...};
Description:	Handle for a specific AdaptivePlatformServiceInstance. It contains the information that is needed to create a service proxy. HandleType satisfies the [equalitycomparable] and [lessthancomparable] requirements as per [31]. This, together with [SWS_CM_00317] and [SWS_CM_00318], allows storing and managing HandleTypes by the application.

⌋

8.2.9.1 Public Member Functions

8.2.9.1.1 Special Member Functions

8.2.9.1.1.1 Copy Constructor

[SWS_CM_00317] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::HandleType`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType	
Syntax:	HandleType (const {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType &other);	
Parameters (in):	other	Other object to copy

▽

△

Exception Safety:	not exception safe
Thread Safety:	implementation defined
Description:	Copy constructor

⌋

8.2.9.1.1.2 Move Constructor

[SWS_CM_00318] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::HandleType`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType	
Syntax:	HandleType ({<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType &&other);	
Parameters (in):	other	Other object to move
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor	

⌋

8.2.9.1.1.3 Default Constructor

[SWS_CM_00349] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::HandleType`

Upstream requirements: RS_AP_00146

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"

▽

△

Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType`
Syntax:	`HandleType ()=delete;`
Description:	Default constructor deletion

⌋

8.2.9.1.1.4 Copy Assignment Operator

[SWS_CM_11532] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::operator=`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_proxy.h"	
Scope:	`class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType`	
Syntax:	`HandleType & operator= (const {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType &other);`	
Parameters (in):	other	Other object to copy
Return value:	HandleType &	New `HandleType`
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Copy assignment constructor	

⌋

8.2.9.1.1.5 Move Assignment Operator

[SWS_CM_11533] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::operator=`

Upstream requirements: RS_CM_00102, RS_AP_00114, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType	
Syntax:	HandleType & operator= ({<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType &&other);	
Parameters (in):	other	Other object to move
Return value:	HandleType &	New HandleType
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Move assignment constructor	

⌋

8.2.9.1.1.6 Destructor

[SWS_CM_11371] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::~HandleType`

Upstream requirements: RS_AP_00114, RS_AP_00132, RS_AP_00145

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType
Syntax:	~HandleType () noexcept;
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor

⌋

8.2.9.1.2 Member Functions

8.2.9.1.2.1 GetInstanceId

[SWS_CM_11531] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::GetInstanceId` ⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType	
Syntax:	const ara::com::InstanceIdentifier & GetInstanceId () const;	
Return value:	const ara::com::InstanceIdentifier &	An ara::com::InstanceIdentifier
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Return the instance identifer for the handle	

⌋

8.2.9.1.2.2 operator<

[SWS_CM_11530] Definition of API function `{<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::operator<` ⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType	
Syntax:	bool operator< (const {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType &other) const;	
Parameters (in):	other	Other HandleType to compare
Return value:	bool	TRUE if other is less than *this, FALSE otherwise.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	LessThan operator as per [lessthancomparable] in [31]	

⌋

8.2.9.1.2.3 operator==

[SWS_CM_11529] Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::operator== ⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_proxy.h"	
Scope:	class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType	
Syntax:	bool operator== (const {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType &other) const;	
Parameters (in):	other	Other HandleType to compare
Return value:	bool	TRUE if other is equal to *this, FALSE otherwise.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Equality operator as per [equalitycomparable] in [31]	

⌋

8.3 Header: {<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h

[SWS_CM_01002] Service Skeleton Header File: file name, includes and multiple inclusion guard

Upstream requirements: RS_CM_00001, RS_AP_00114, RS_AP_00116

⌈

Kind:	Header File	
Syntax:	{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h	
Description:	The Service Skeleton Header File is the provided (server side) API interface of a ServiceInterface. It contains different classes representing the ServiceInterface itself and its elements event, method, field and trigger. For each modeled ServiceInterface a Service Skeleton Header File is generated according to this directory path/file name convention and shall: 1. Insert a multiple inclusion guard around the whole header file as per [SWS_CORE_90002] 2. Include the Service Common Header File as per [SWS_CM_01012]	
Descriptors:	{<si-namespace-derived-directory-path-lower>}	as per {<si-namespace-derived-directory-path-lower>} in [SWS_CM_01012]

▽

△

	{<si-shortname-lower>}	as per {<si-shortname-lower>} in [SWS_CM_01012]
Example:	// File=n/n_plus_1/n_plus_2/si_skeleton.h (1) #ifndef N_NPLUS1_NPLUS2_SI_SKELETON_H_ (2) #define N_NPLUS1_NPLUS2_SI_SKELETON_H_ (2) #include ".../path/to/si_common.h" (3) ... #endif // N_NPLUS1_NPLUS2_SI_SKELETON_H_ (2)	

⌋

8.3.1 Namespaces

8.3.1.1 {<hierarchical-namespace-list-lower-skeleton>}

[SWS_CM_01005] Service Skeleton Header File: service namespace

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	--
Syntax:	namespace {<hierarchical-namespace-list-lower-skeleton>}
Description:	The generator shall use the SymbolProps aggregated in the role PortInterface.namespace to construct the encapsulating C++ namespace hierarchy as per [SWS_CM_11500].

⌋

8.3.1.2 {<hierarchical-namespace-list-lower-skeleton>}::skeleton

[SWS_CM_01006] Service Skeleton Header File: skeleton namespace

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"

▽

△

Scope:	`namespace {<hierarchical-namespace-list-lower-skeleton>}`
Syntax:	`namespace skeleton`
Description:	Inner namespace for definitions on skeleton level.

⌋

8.3.1.3 {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events

[SWS_CM_01009] `Service Skeleton Header File`**: events namespace**

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"
Scope:	`namespace {<hierarchical-namespace-list-lower-skeleton>}::skeleton`
Syntax:	`namespace events`
Description:	Inner namespace for definitions on events level.

⌋

8.3.1.4 {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields

[SWS_CM_01031] `Service Skeleton Header File`**: fields namespace**

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"
Scope:	`namespace {<hierarchical-namespace-list-lower-skeleton>}::skeleton`
Syntax:	`namespace fields`
Description:	Inner namespace for definitions on fields level.

⌋

8.3.1.5 {<hierarchical-namespace-list-lower-skeleton>}::skeleton::triggers

[SWS_CM_11502] `Service Skeleton Header File`: triggers namespace

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	namespace {<hierarchical-namespace-list-lower-skeleton>}::skeleton
Syntax:	namespace triggers
Description:	Inner namespace for definitions on triggers level.

⌋

8.3.2 Class: {<event-name-upper-camel>}

[SWS_CM_00003] Service Skeleton event class

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton_fwd.h"	
Scope:	namespace {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events	
Symbol:	{<event-name-upper-camel>}	
Syntax:	class {<event-name-upper-camel>} {...};	
Description:	Service Skeleton event class, one for each VariableDataPrototype defined in the ServiceInterface in the role event.	
Descriptors:	{<event-name-upper-camel>}	Name of an event created from the VariableDataPrototype. shortName defined in the role ServiceInterface. event converted to upper camel-case letters.

⌋

8.3.2.1 Public Member Types

8.3.2.1.1 Type Alias: SampleType

[SWS_CM_11400] Definition of API type {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::SampleType

Upstream requirements: RS_CM_00201, RS_AP_00114

⌈

Kind:	type alias	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}	
Symbol:	SampleType	
Syntax:	using SampleType = {<s-sample-cppidt-symbol>};	
Description:	Data type for the value of an event sample (the CppImplementationDataType of the Event, i.e. for the CppImplementationDataType which is either referenced by the VariableDataPrototype in the role type directly or via the indirection of a DataTypeMap)	
Descriptors:	{<s-sample-cppidt-symbol>}	The CppImplementationDataType. shortName of the CppImplementationDataType of the VariableDataPrototype

⌋

8.3.2.2 Public Member Functions

8.3.2.2.1 Member Functions

8.3.2.2.1.1 Allocate

[SWS_CM_90438] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::Allocate

Upstream requirements: RS_CM_00201, RS_AP_00114, RS_AP_00115, RS_AP_00120

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}
Syntax:	ara::com::SampleAllocateePtr< SampleType > Allocate ();

▽

△

Return value:	ara::com::Sample AllocateePtr< Sample Type >	A `ara::com::SampleAllocateePtr` to an event sample
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Violations:	`SampleAllocation-Violation`	Not enough memory resources can be allocated
	`AllocateUsageVio-lation`	The allocation was illegally done via custom allocator
Description:	Allocating data for event transfer when Communication Management is responsible for the data.	

⌋

8.3.2.2.1.2 SetSubscriptionStateChangeHandler

[SWS_CM_12008] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::SetSubscriptionStateChangeHandler` ⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"	
Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}`	
Syntax:	`void SetSubscriptionStateChangeHandler (ara::com::SubscriptionStateChangeHandler handler) noexcept;`	
Parameters (in):	handler	The handler (as per [SWS_CM_00311]) that shall be called by Communication Management as soon as the `ara::com::SubscriptionState` of the event has changed.
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerVio-lation`	Provided `Callable` handler does not exist or is `null`
Description:	Set/Register `ara::com::SubscriptionState` change handler	

⌋

8.3.2.2.2 Service Communication

8.3.2.2.2.1 Send

[SWS_CM_00162] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::Send

Upstream requirements: RS_CM_00201, RS_AP_00114, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}	
Syntax:	ara::core::Result< void > Send ({<s-sample-derived-type>} data) noexcept;	
Parameters (in):	data	The data to send out to the subscribed applications.
Return value:	ara::core::Result< void >	• If successful: an ara::core::Result containing a ara::core::Result::value_type containing a void • If unsuccessful: an ara::core::Result containing an ara::core::Result::error_type i.e. a corresponding ara::com::ComErrc
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kServiceNotOffered	rollback_semantics Service has not yet been offered via OfferService()
Description:	Send event where the application is responsible for the data. The Communication Management creates a copy of the data to send and sends it to all subscribing applications.	
Descriptors:	{<s-sample-derived-type>}	As per {<derived-type>} in [SWS_CM_00191]

⌋

8.3.2.2.2 Send

[SWS_CM_90437] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::Send

Upstream requirements: RS_CM_00201, RS_AP_00114, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}	
Syntax:	ara::core::Result< void > Send (ara::com::SampleAllocateePtr< SampleType > data) noexcept;	
Parameters (in):	data	the data to send.
Return value:	ara::core::Result< void >	• If successful: an ara::core::Result containing a ara::core::Result::value_type containing a void • If unsuccessful: an ara::core::Result containing an ara::core::Result::error_type i.e. a corresponding ara::com::ComErrc
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kServiceNotOffered	rollback_semantics
		Service has not yet been offered via OfferService()
Description:	Send event where Communication Management is responsible for the data. The data is allocated using Allocate(). The application is not allowed to access the data after Communication Management sends it to all subscribing applications.	

⌋

8.3.2.2.3 Service Discovery

8.3.2.2.3.1 GetSubscriptionState

[SWS_CM_12011] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::GetSubscriptionState ⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}
Syntax:	ara::com::SubscriptionState GetSubscriptionState () const noexcept;

▽

△

Return value:	ara::com::Subscription State	The state of the subscription (as per [SWS_CM_12008]).
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Query `ara::com::SubscriptionState` of an event.	

⌋

8.3.2.2.3.2 SetSubscriptionStateChangeHandler

[SWS_CM_12009] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}`::skeleton::events::`{<event-name-upper-camel>}`::SetSubscriptionStateChangeHandler ⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"	
Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}`	
Syntax:	`template <typename ExecutorT> void SetSubscriptionStateChangeHandler (ara::com::SubscriptionStateChangeHandler handler, ExecutorT &&executor) noexcept;`	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	handler	As per [SWS_CM_12008]
	executor	As per [SWS_CM_11360]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_12008] but the method shall execute in a provided context	

⌋

8.3.2.2.3.3 UnsetSubscriptionStateChangeHandler

[SWS_CM_12010] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::UnsetSubscriptionStateChangeHandler` ⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}
Syntax:	void UnsetSubscriptionStateChangeHandler () noexcept;
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Unset/Deregister the ara::com::SubscriptionState change handler

⌋

8.3.3 Class: {<field-name-upper-camel>}

[SWS_CM_00007] Definition of API class `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}`

Upstream requirements: RS_CM_00219, RS_AP_00114

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton_fwd.h"	
Scope:	namespace {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields	
Symbol:	{<field-name-upper-camel>}	
Syntax:	class {<field-name-upper-camel>} {...};	
Description:	Service skeleton Field class, one for each Field defined in the ServiceInterface in the role field.	
Descriptors:	{<field-name-upper-camel>}	Name of a field created from the Field.shortName defined in the ServiceInterface. field converted to upper camel-case letters.

⌋

8.3.3.1 Public Member Types

8.3.3.1.1 Type Alias: FieldType

[SWS_CM_11402] Definition of API type {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::Field Type

Upstream requirements: RS_CM_00219, RS_AP_00114

⌈

Kind:	type alias	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}	
Symbol:	FieldType	
Syntax:	using FieldType = {<s-field-cppidt-symbol>};	
Description:	Data type for the value of a field (an alias for the CppImplementationDataType of the Field, i.e. for the CppImplementationDataType which is either referenced by the VariableDataPrototype in the role type directly or via indirection of a DataTypeMap)	
Descriptors:	{<s-field-cppidt-symbol>}	The CppImplementationDataType. shortName of the Field

⌋

8.3.3.2 Public Member Functions

8.3.3.2.1 Member Functions

8.3.3.2.1.1 SetTransportFaultConditionHandler

[SWS_CM_00097] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::SetTransportFaultConditionHandler

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}

▽

△

Syntax:	`template <typename ExecutorT>` `void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler, ExecutorT &&executor) noexcept;`	
Parameters (in):	handler	Handler of type TransportFaultConditionHandler to be set
	executor	Executor object in which any asynchronous computation spawn shall be invoked
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Execution Context overload for SetTransportFaultConditionHandler for a given Skeleton Field Get/Set.	
See also:	[SWS_CM_00096]	

⌋

8.3.3.2.1.2 SetTransportFaultConditionHandler

[SWS_CM_00096] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::SetTransportFaultConditionHandler`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"	
Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}`	
Syntax:	`void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler) noexcept;`	
Parameters (in):	handler	Handler of type TransportFaultConditionHandler to be set
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Provides possibility to register a TransportFaultConditionHandler for a given Skeleton Field Get/Set.	

⌋

8.3.3.2.1.3 UnsetTransportFaultConditionHandler

[SWS_CM_00098] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::UnsetTransportFaultConditionHandler`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"
Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}`
Syntax:	`void UnsetTransportFaultConditionHandler () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Provides the possibility to unregister a previously set TransportFaultConditionHandler for a given Skeleton Field Get/Set.

⌋

8.3.3.2.2 Service Communication

8.3.3.2.2.1 Update

[SWS_CM_00119] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::Update`

Upstream requirements: RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"	
Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}`	
Syntax:	`void Update ({<s-field-derived-type>} value) noexcept;`	
Parameters (in):	value	The value to be sent to subscribers
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	

▽

△

Description:	Initiate the transmission of updated field data to the subscribers. See [SWS_CM_00162] for the required behavior. The Update function shall also update the fields internal value. An update notification shall be sent to any subscribers if: • `Field.hasNotifier ==TRUE` (according to [SWS_CM_00120])	
Descriptors:	`{<s-field-derived-type>}`	As per `{<derived-type>}` in [SWS_CM_00191]

⌋

8.3.3.2.3 Service Management

8.3.3.2.3.1 RegisterGetHandler

[SWS_CM_11360] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::RegisterGetHandler`

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"	
Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}`	
Syntax:	`template <typename ExecutorT> void RegisterGetHandler (std::function< ara::core::Future< FieldType >()> handler, ExecutorT &&executor) noexcept;`	
Template param:	ExecutorT	Wrapper type providing methods for asynchronous execution of `Callable`s. AUTOSAR provides a standardized `ara::core::Executor` or implementations may use a custom type which provides the same interface.
Parameters (in):	handler	As per [SWS_CM_00114]
	executor	The context in which this method shall be "executed"
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_00114] but the method shall execute in a provided context	

⌋

8.3.3.2.3.2 RegisterGetHandler

[SWS_CM_00114] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::RegisterGetHandler`

Upstream requirements: RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}	
Syntax:	void RegisterGetHandler (std::function< ara::core::Future< FieldType >()> handler) noexcept;	
Parameters (in):	handler	The callback function to be invoked to process the Field get request
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InvalidHandlerViolation	Provided Callable handler does not exist or is null
Description:	Register a callback function to be invoked when a Get() request is received. The method exists only if the offered service contains a Field and Field.hasGetter ==TRUE	

⌋

8.3.3.2.3.3 RegisterSetHandler

[SWS_CM_11362] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::RegisterSetHandler`

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}

▽

△

Syntax:	`template <typename ExecutorT> void RegisterSetHandler (std::function< ara::core::Future< FieldType >({<s-field-derived-type>} value)> setHandler, ExecutorT &&executor) noexcept;`	
Template param:	ExecutorT	As per [SWS_CM_11360]
Parameters (in):	setHandler	As per [SWS_CM_00116]
	executor	As per [SWS_CM_11360]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	As per [SWS_CM_00116] but the method shall execute in a provided context	
Descriptors:	{<s-field-derived-type>}	As per {<derived-type>} in [SWS_CM_00191]

⌋

8.3.3.2.3.4 RegisterSetHandler

[SWS_CM_00116] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::RegisterSetHandler

Upstream requirements: RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}	
Syntax:	`void RegisterSetHandler (std::function< ara::core::Future< FieldType >({<s-field-derived-type>} value)> handler) noexcept;`	
Parameters (in):	handler	As per [SWS_CM_00114]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Register a callback function to be invoked when a `Set()` request is received. The method exists only if the offered service contains a `Field` and `Field.hasSetter` ==TRUE	
Descriptors:	{<s-field-derived-type>}	As per {<derived-type>} in [SWS_CM_00191]

⌋

8.3.4 Class: {<trigger-name-upper-camel>}

[SWS_CM_00726] Definition of API class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::triggers::{<trigger-name-upper-camel>}

Upstream requirements: RS_CM_00201, RS_AP_00114

⌈

Kind:	class	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton_fwd.h"	
Scope:	namespace {<hierarchical-namespace-list-lower-skeleton>}::skeleton::triggers	
Symbol:	{<trigger-name-upper-camel>}	
Syntax:	class {<trigger-name-upper-camel>} {...};	
Description:	Service skeleton Trigger class, one for each Trigger defined in the ServiceInterface in the role trigger.	
Descriptors:	{<trigger-name-upper-camel>}	Name of a trigger created from the Trigger.shortName defined in the ServiceInterface in the role trigger converted to upper camel-case letters.

⌋

8.3.4.1 Public Member Functions

8.3.4.1.1 Member Functions

8.3.4.1.1.1 Send

[SWS_CM_00721] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::triggers::{<trigger-name-upper-camel>}::Send

Upstream requirements: RS_CM_00201, RS_AP_00114, RS_AP_00121

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::triggers::{<trigger-name-upper-camel>}
Syntax:	ara::core::Result< void > Send () noexcept;

▽

△

Return value:	ara::core::Result< void >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `void` • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::com::ComErrc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kServiceNotOffered	rollback_semantics
		Service has not yet been offered via `OfferService()`
Description:	Send trigger to all subscribing applications.	

⌋

8.3.5 Class: {<service-interface-name-upper-camel>}Skeleton

[SWS_CM_00002] Definition of API class `{<hierarchical-namespace-list-lower-skeleton>}`::skeleton::`{<service-interface-name-upper-camel>}`Skeleton

Upstream requirements: RS_CM_00101, RS_AP_00114, RS_AP_00122

⌈

Kind:	class	
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"	
Forwarding header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton_fwd.h"	
Scope:	`namespace {<hierarchical-namespace-list-lower-skeleton>}::skeleton`	
Symbol:	`{<service-interface-name-upper-camel>}`Skeleton	
Syntax:	`class {<service-interface-name-upper-camel>}Skeleton {...};`	
Description:	Service skeleton class	
Descriptors:	`{<service-interface-name-upper-camel>}`	The `ServiceInterface`. `shortName` converted to upper camel-case letters

⌋

8.3.5.1 Public Member Variables

8.3.5.1.1 {<event-name-upper-camel>}

[SWS_CM_99557] Definition of API variable {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<event-name-upper-camel>}

Upstream requirements: RS_CM_00201, RS_AP_00114

⌈

Kind:	variable	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Symbol:	{<event-name-upper-camel>}	
Type:	events::{<event-name-upper-camel>}	
Syntax:	events::{<event-name-upper-camel>} {<event-name-upper-camel>};	
Description:	Event member, one for each VariableDataPrototype defined in the ServiceInterface in the role event.	
Descriptors:	{<event-name-upper-camel>}	As per {<event-name-upper-camel>} in [SWS_CM_00003]

⌋

8.3.5.1.2 {<field-name-upper-camel>}

[SWS_CM_99558] Definition of API variable {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<field-name-upper-camel>}

Upstream requirements: RS_CM_00216, RS_AP_00114

⌈

Kind:	variable
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton
Symbol:	{<field-name-upper-camel>}
Type:	fields::{<field-name-upper-camel>}
Syntax:	fields::{<field-name-upper-camel>} {<field-name-upper-camel>};
Description:	Field member, one for each Field defined in the ServiceInterface in the role field.

▽

△

Descriptors:	{<field-name-upper-camel>}	As per {<field-name-upper-camel>} in [SWS_CM_00007]

⌋

8.3.5.1.3 {<method-out-arg-symbol>}

[SWS_CM_11550] Definition of API variable {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<method-out-arg-symbol>}

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	variable	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Symbol:	{<method-out-arg-symbol>}	
Type:	{<method-out-arg-type>}	
Syntax:	{<method-out-arg-type>} {<method-out-arg-symbol>};	
Description:	Member declaration representing an *out* argument in an Output.	
Descriptors:	{<method-out-arg-type>}	The ClientServerOperation. argument. type, mapped to a C++ data type according to [30].
	{<method-out-arg-symbol>}	Symbol name of the struct element as given by ClientServerOperation. ArgumentDataPrototype. shortName

⌋

8.3.5.1.4 {<trigger-name-upper-camel>}

[SWS_CM_99559] Definition of API variable {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<trigger-name-upper-camel>}

Upstream requirements: RS_CM_00201, RS_AP_00114

⌈

Kind:	variable	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Symbol:	{<trigger-name-upper-camel>}	
Type:	triggers::{<trigger-name-upper-camel>}	
Syntax:	triggers::{<trigger-name-upper-camel>} {<trigger-name-upper-camel>};	
Description:	Trigger member, one for each Trigger defined in the ServiceInterface in the role trigger.	
Descriptors:	{<trigger-name-upper-camel>}	As per {<trigger-name-upper-camel>} in [SWS_CM_00726]

⌋

8.3.5.2 Public Member Functions

8.3.5.2.1 Special Member Functions

8.3.5.2.1.1 Move Assignment Operator

[SWS_CM_11549] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::operator=

Upstream requirements: RS_CM_00101, RS_AP_00114

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	{<service-interface-name-upper-camel>}Skeleton & operator= ({<service-interface-name-upper-camel>}Skeleton &&other) noexcept;	
Parameters (in):	other	Other object to move

▽

△

Return value:	{<service-interface-name-upper-camel>}Skeleton &	The moved skeleton object
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move assignment constructor	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.3.5.2.1.2 Copy Assignment Operator

[SWS_CM_11548] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::operator=

Upstream requirements: RS_CM_00101, RS_AP_00114

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	{<service-interface-name-upper-camel>}Skeleton & operator= (const {<service-interface-name-upper-camel>}Skeleton &other)=delete;	
Description:	Copy assignment constructor deletion	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.3.5.2.1.3 Move Constructor

[SWS_CM_00135] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton

Upstream requirements: RS_CM_00101, RS_AP_00114

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	{<service-interface-name-upper-camel>}Skeleton ({<service-interface-name-upper-camel>}Skeleton &&other);	
Parameters (in):	other	Other object to move
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.3.5.2.1.4 Copy Constructor

[SWS_CM_00134] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton

Upstream requirements: RS_CM_00101, RS_AP_00114

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton
Syntax:	{<service-interface-name-upper-camel>}Skeleton (const {<service-interface-name-upper-camel>}Skeleton &other)=delete;
Description:	Copy constructor deletion

▽

△

Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.3.5.2.1.5 Destructor

[SWS_CM_11370] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::~{<service-interface-name-upper-camel>}Skeleton

Upstream requirements: RS_AP_00145

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	virtual ~{<service-interface-name-upper-camel>}Skeleton ();	
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Violations:	DanglingSamplesViolation	Precondition [SWS_CM_00086] has been violated.
Description:	Destruction of a Skeleton	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.3.5.2.2 Constructors

8.3.5.2.2.1 {<service-interface-name-upper-camel>}Skeleton

[SWS_CM_00153] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton

Upstream requirements: RS_CM_00101, RS_AP_00114, RS_AP_00115, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	{<service-interface-name-upper-camel>}Skeleton (ara::com::InstanceIdentifierContainer instanceIDs, ara::com::MethodCallProcessingMode mode=ara::com::MethodCallProcessingMode::kEvent) noexcept;	
Parameters (in):	instanceIDs	A container of instance identifiers of a service.
	mode	Mode (as per [SWS_CM_00301]) for processing incoming service method invocations. Default argument is kEvent.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InsufficientPermissionsViolation	Refused by Grant Enforcement
	WrongMethodCallProcessingModeViolation	Wrong processing mode passed to constructor
Description:	Construct a Skeleton from an ara::com::InstanceIdentifierContainer	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.3.5.2.2.2 {<service-interface-name-upper-camel>}Skeleton

[SWS_CM_00152] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton

Upstream requirements: RS_CM_00101, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00127, RS_AP_00137

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	{<service-interface-name-upper-camel>}Skeleton (ara::core::InstanceSpecifier instanceSpec, ara::com::MethodCallProcessingMode mode=ara::com::MethodCallProcessingMode::kEvent) noexcept;	
Parameters (in):	instanceSpec	The specifier of a specific instance of a service.
	mode	Mode (as per [SWS_CM_00301]) for processing incoming service method invocations. Default argument is kEvent.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InstanceSpecifierMappingIntegrityViolation	InstanceSpecifier either cannot be resolved in the model in the context of the executable, or it refers to a model element other than a PortPrototype.
	PortInterfaceMappingViolation	The type of mapping does not match the expected type of Port Interface: ServiceInterface referenced by a ServiceInstanceToPortPrototypeMapping.
	ProcessMappingViolation	The type of mapping does not match the expected type of Port Interface
	InstanceSpecifierAlreadyInUseViolation	The constructor was called with an Instance Specifier already
	WrongMethodCallProcessingModeViolation	Wrong processing mode passed to constructor
Description:	Construct a Skeleton from an ara::core::InstanceSpecifier	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]

⌋

8.3.5.2.2.3 {<service-interface-name-upper-camel>}Skeleton

[SWS_CM_00130] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton

Upstream requirements: RS_CM_00101, RS_AP_00114, RS_AP_00121

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	{<service-interface-name-upper-camel>}Skeleton (ara::com::InstanceIdentifier instanceID, ara::com::MethodCallProcessingMode mode=ara::com::MethodCallProcessingMode::kEvent) noexcept;	
Parameters (in):	instanceID	The identifier of a specific instance of a service.
	mode	Mode (as per [SWS_CM_00301]) for processing incoming service method invocations. Default argument is kEvent.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InsufficientPermissionsViolation	Refused by Grant Enforcement
	WrongMethodCallProcessingModeViolation	Wrong processing mode passed to constructor
Description:	Construct a Skeleton from an ara::com::InstanceIdentifier	
Descriptors:	{<service-interface-name-upper-camel>}	As per {<service-interface-name-upper-camel>} in [SWS_CM_00002]
See also:	[SWS_CM_00301], [SWS_CM_00302]	

⌋

8.3.5.2.3 Member Functions

8.3.5.2.3.1 SetTransportFaultConditionHandler

[SWS_CM_00094] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::SetTransportFaultConditionHandler`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	template <typename ExecutorT> void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler, ExecutorT &&executor) noexcept;	
Parameters (in):	handler	Handler of type TransportFaultConditionHandler to be set
	executor	Executor object in which any asynchronous computation spawn shall be invoked
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	InvalidHandlerViolation	Provided Callable handler does not exist or is null
Description:	Execution Context overload for SetTransportFaultConditionHandler for all methods in a given Skeleton.	
See also:	[SWS_CM_00093]	

⌋

8.3.5.2.3.2 SetTransportFaultConditionHandler

[SWS_CM_00093] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::SetTransportFaultConditionHandler`

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton

▽

△

Syntax:	`void SetTransportFaultConditionHandler (ara::com::e2e::TransportFaultConditionHandler handler) noexcept;`	
Parameters (in):	handler	Handler of type TransportFaultConditionHandler to be set
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidHandlerViolation`	Provided `Callable` handler does not exist or is `null`
Description:	Provides possibility to register a TransportFaultConditionHandler for all methods in a given Skeleton.	

⌋

8.3.5.2.3.3 UnsetTransportFaultConditionHandler

[SWS_CM_00095] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}`::skeleton::`{<service-interface-name-upper-camel>}`Skeleton::UnsetTransportFaultConditionHandler

Upstream requirements: RS_CM_00224, RS_E2E_08534

⌈

Kind:	function
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"
Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton`
Syntax:	`void UnsetTransportFaultConditionHandler () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Provides the possibility to unregister a previously set TransportFaultConditionHandler for all methods in a given Skeleton.

⌋

8.3.5.2.3.4 {<fnfmethod-name-upper-camel>}

[SWS_CM_90434] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<fnfmethod-name-upper-camel>}

Upstream requirements: RS_CM_00215, RS_AP_00114

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	virtual void {<fnfmethod-name-upper-camel>} ({<fnfmethod-in-arg-derived-type-0toN>} {<fnfmethod-in-arg-symbol-0toN>}) noexcept=0;	
Parameters (in):	{<fnfmethod-in-arg-symbol-0toN>}	As per {<method-in-arg-symbol-0toN>} in [SWS_CM_00191]
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Provision of a fire-and-forget method (ClientServerOperation. fireAndForget ==TRUE)	
Descriptors:	{<fnfmethod-name-upper-camel>}	As per {<method-name-upper-camel>} in [SWS_CM_00191]
	{<fnfmethod-in-arg-derived-type-0toN>}	As per {<method-in-arg-derived-type-0toN>} in [SWS_CM_00191]

⌋

8.3.5.2.4 Service Communication

8.3.5.2.4.1 {<method-name-upper-camel>}

[SWS_CM_00191] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<method-name-upper-camel>}

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	virtual ara::core::Future< {<method-name-upper-camel>}Output > {<method-name-upper-camel>} ({<method-in-arg-derived-type-0toN>} {<method-in-arg-symbol-0toN>}) noexcept=0;	
Parameters (in):	{<method-in-arg-symbol-0toN>}	The symbol name of method argument[0..N] in the ordered list of method arguments, with ClientServerOperation. direction == in **or** direction == inout given by ClientServerOperation. ArgumentDataPrototype[0..N]. shortName
Return value:	ara::core::Future< {<method-name-upper-camel>}Output >	• If successful: an ara::core::Future containing an Output object as per [SWS_CM_99556] • If unsuccessful: an ara::core::Future containing a corresponding ara::com::ComErrc or ApApplicationError.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Provision of a method (ClientServerOperation. fireAndForget ==FALSE)	
Descriptors:	{<method-name-upper-camel>}	Name of a method created from the ClientServerOperation. shortName defined in the ServiceInterface in the role method converted to upper camel-case letters.
	{<derived-type>}	The C++ data type shall be derived according to the following rules: 1. If the (CppImplementationDataType. category ==VALUE **or** a (CppImplementationDataType. category ==TYPE_REFERENCE which transitively type-resolves to a CppImplementationDataType. category ==VALUE) **and** the CppImplementationDataType. shortName is either: • int8_t: as per [SWS_APT_00001] • int16_t: as per [SWS_APT_00004] • int32_t: as per [SWS_APT_00007] • int64_t: as per [SWS_APT_00010] • uint8_t: as per [SWS_APT_00022] • uint16_t: as per [SWS_APT_00025] • uint32_t: as per [SWS_APT_00028]

▽

▽

△

		△ • uint64_t: as per [SWS_APT_00031] • bool: as per [SWS_APT_00049] • float: as per [SWS_APT_00043] • double: as per [SWS_APT_00046] in [15], then {<derived-type>} shall be unchanged, i.e. shall be the mapped C++ data type according to [30]. 2. Otherwise {<derived-type>} shall be a const-qualified lvalue reference (see [basic.type.qualifier], [basic.lval] in [31]), to the mapped C++ data type according to [30].
	{<method-in-arg-derived-type-0toN>}	The data type of method argument[0..N] in the ordered list of method arguments, with ClientServerOperation. direction == in **or** direction == inout given by ClientServerOperation. argument[0..N]. type, derived as per {<derived-type>}
Example:		

```
// Example:
virtual ara::core::Future<SomeMethodOutput>
SomeMethodOutput(
  // {<derived-type>} sub-clause 1:
  std::int8_t inArg0,

  // {<derived-type>} sub-clause 1:
  //     e.g. if inArg1 would type-resolve to std::uint64_t
  ns1::ns2::sometype_t inArg1,

  // {<derived-type>} sub-clause 2:
  //     e.g. if inArg2 would type-resolve to ara::core::map
  const ns1::ns2::anothertype_t& inArg2,

  // {<derived-type>} sub-clause 2:
  //     e.g. if inArgN would type-resolve to struct
  const ns1::ns2::furthertype_t& inArgN
) = 0;
```

⌋

8.3.5.2.5 Service Discovery

8.3.5.2.5.1 OfferService

[SWS_CM_00101] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::OfferService

Upstream requirements: RS_CM_00101, RS_AP_00114, RS_AP_00120

⌈

Kind:	function
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"

▽

△

Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton`	
Syntax:	`ara::core::Result< void > OfferService () noexcept;`	
Return value:	ara::core::Result< void >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::com::ComErrc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kFieldValue NotInitialized	rollback_semantics
		Field value has yet not been initialized via `Update()` (see [SWS_CM_00128]
	ComErrc::kFieldSet HandlerNotSet	rollback_semantics
		Field SetHandler has not been registered via `RegisterSetHandler()`
	ComErrc::kNetwork BindingFailure	rollback_semantics
		The network binding reported a recoverable communications error or a secure communication failure
Description:	Method to offer a service	

⌋

8.3.5.2.5.2 StopOfferService

[SWS_CM_00111] Definition of API function `{<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::StopOfferService`

Upstream requirements: RS_CM_00105, RS_AP_00114, RS_AP_00120

⌈

Kind:	function
Header file:	#include "`{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}`_skeleton.h"
Scope:	`class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton`
Syntax:	`void StopOfferService () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Method to stop offering a service

⌋

8.3.5.2.6 Service Management

8.3.5.2.6.1 ProcessNextMethodCall

[SWS_CM_00199] Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::ProcessNextMethodCall

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	function	
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Syntax:	bool ProcessNextMethodCall () noexcept;	
Return value:	bool	If the bool type: • ==TRUE: there is at least one pending invocation • ==FALSE: there is no pending invocation
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	WrongMethodCallProcessingModeViolation	The method is called in a Skeleton which was constructed with ara::com::MethodCallProcessingMode == kEvent or ara::com::MethodCallProcessingMode == kEventSingleThread.
Description:	Allows the service to trigger the execution of the next service consumer method call at a specific point of time if the ara::com::MethodCallProcessingMode == kPolling The pending service method (see [SWS_CM_00191] or [SWS_CM_90434]) is executed in the context of the call to ProcessNextMethodCall().	

⌋

8.3.6 Struct: {<method-name-upper-camel>}Output

[SWS_CM_99556] Definition of API class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<method-name-upper-camel>}Output

Upstream requirements: RS_CM_00211, RS_AP_00114, RS_AP_00127, RS_AP_00128, RS_AP_00138

⌈

Kind:	struct
Header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton.h"

▽

△

Forwarding header file:	#include "{<si-namespace-derived-directory-path-lower>}/{<si-shortname-lower>}_skeleton_fwd.h"	
Scope:	class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton	
Symbol:	{<method-name-upper-camel>}Output	
Syntax:	struct {<method-name-upper-camel>}Output {...};	
Description:	Structure wrapping the *out* arguments for a method	
Descriptors:	{<method-name-upper-camel>}	As per {<method-name-upper-camel>} in [SWS_CM_00191]
	{<method-out-args-members-ordered>}	**Shown as "..." in Syntax**. Each element is an argument in the ordered list of arguments, with direction == out **or** direction == inout, where [SWS_CM_11550] applies for each.

⌋

8.4 Header: ara/com/types.h

[SWS_CM_01013] ara::com Types Header File: file name

Upstream requirements: RS_CM_00001, RS_AP_00114, RS_AP_00116

⌈

Kind:	Header File
Syntax:	ara/com/types.h
Description:	The ara::com Types Header File provides the core ara::com API interface. In particular the data types used in the Service Skeleton Header File and Service Proxy Header File definitions

⌋

8.4.1 Non-Member Types

8.4.1.1 Type Alias: EventReceiveHandler

[SWS_CM_00309] Definition of API type ara::com::EventReceiveHandler

Upstream requirements: RS_CM_00203, RS_AP_00114, RS_AP_00120

⌈

Kind:	type alias
Header file:	#include "ara/com/types.h"
Scope:	`namespace ara::com`
Symbol:	EventReceiveHandler
Syntax:	`using EventReceiveHandler = std::function<void()>;`
Thread Safety:	implementation defined
Description:	Callback function invoked if new event data arrives for an event. The event receiver must provide the function implementation which is not required to be re-entrant. Usage of `std::function` is recommended but not mandatory.

⌋

8.4.1.2 Type Alias: FieldReceiveHandler

[SWS_CM_11615] Definition of API type ara::com::FieldReceiveHandler

Upstream requirements: RS_CM_00203, RS_AP_00114, RS_AP_00120

⌈

Kind:	type alias
Header file:	#include "ara/com/types.h"
Scope:	`namespace ara::com`
Symbol:	FieldReceiveHandler
Syntax:	`using FieldReceiveHandler = std::function<void()>;`
Thread Safety:	implementation defined
Description:	Callback function invoked if a new notification arrives for a field. The field receiver must provide the function implementation which is not required to be re-entrant. Usage of `std::function` is recommended but not mandatory.

⌋

8.4.1.3 Type Alias: FindServiceHandler

[SWS_CM_00383] Definition of API type ara::com::FindServiceHandler

Upstream requirements: RS_CM_00102

⌈

Kind:	type alias	
Header file:	#include "ara/com/types.h"	
Scope:	`namespace ara::com`	
Symbol:	FindServiceHandler	
Syntax:	`using FindServiceHandler = std::function<void(ServiceHandle Container<T>, FindServiceHandle)>;`	
Template param:	T	Data type of data sample
Thread Safety:	implementation defined	
Description:	Callback function invoked if service availability changes. It takes as input parameter a handle container containing handles for all matching service instances and a `ara::com::FindServiceHandle` which can be used to invoke `StopFindService()` from within the `ara::com::FindServiceHandler`.	
See also:	[SWS_CM_00125]	

⌋

8.4.1.4 Type Alias: InstanceIdentifierContainer

[SWS_CM_00319] Definition of API type ara::com::InstanceIdentifierContainer

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00122

⌈

Kind:	type alias
Header file:	#include "ara/com/types.h"
Scope:	`namespace ara::com`
Symbol:	InstanceIdentifierContainer
Syntax:	`using InstanceIdentifierContainer = ara::core::Vector<InstanceIdentifier>;`
Description:	Holds a list of `ara::com::InstanceIdentifier`s. The assigned data type is allowed to be changed by the Communication Management software provider, but must adhere to `[container.requirements.general]` and `[sequence.reqmts]` requirements as per [31] A `ara::core::Vector` ([SWS_CORE_01301]) for example fulfills these requirements.

⌋

8.4.1.5 Enumeration: MethodCallProcessingMode

[SWS_CM_00301] Definition of API enum ara::com::MethodCallProcessingMode

Upstream requirements: RS_CM_00211

⌈

Kind:	enumeration	
Header file:	#include "ara/com/types.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com`	
Symbol:	MethodCallProcessingMode	
Underlying type:	std::uint8_t	
Syntax:	`enum class MethodCallProcessingMode : std::uint8_t {...};`	
Values:	kPoll	Polling mode
	kEvent	Event driven and concurrent mode
	kEventSingleThread	Event driven, sequential mode
Description:	Processing modes for the service implementation.	

⌋

8.4.1.6 Type Alias: SampleAllocateePtr

[SWS_CM_00308] Definition of API type ara::com::SampleAllocateePtr

Upstream requirements: RS_CM_00201

⌈

Kind:	type alias	
Header file:	#include "ara/com/types.h"	
Scope:	`namespace ara::com`	
Symbol:	SampleAllocateePtr	
Syntax:	`using SampleAllocateePtr = std::unique_ptr<T>;`	
Template param:	T	Data type of data sample
Description:	Pointer to an allocated (by Communication Management) data sample. The implementation is allowed to be changed by the Communication Management software provider. The precondition defined in [SWS_CM_00086] must be fulfilled. Otherwise it is considered a violation.	

⌋

8.4.1.7 Type Alias: ServiceHandleContainer

[SWS_CM_00304] Definition of API type ara::com::ServiceHandleContainer

Upstream requirements: RS_CM_00102

⌈

Kind:	type alias	
Header file:	#include "ara/com/types.h"	
Scope:	`namespace ara::com`	
Symbol:	ServiceHandleContainer	
Syntax:	`using ServiceHandleContainer = ara::core::Vector<T>;`	
Template param:	T	service handle
Description:	Holds a list of service handles and is used as a return value of **any** of the `FindService()` methods) • [SWS_CM_00122] • [SWS_CM_00622] . The assigned data type is allowed to be changed by the Communication Management software provider, but must adhere to `[container.requirements.general]` and `[sequence.reqmts]` requirements as per [31]. A `ara::core::Vector` ([SWS_CORE_01301]) for example fulfills these requirements.	

⌋

8.4.1.8 Enumeration: ServiceState

[SWS_CM_01071] Definition of API enum ara::com::ServiceState ⌈

Kind:	enumeration	
Header file:	#include "ara/com/types.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com`	
Symbol:	ServiceState	
Underlying type:	std::uint8_t	
Syntax:	`enum class ServiceState : std::uint8_t {...};`	
Values:	kNotAvailable= 0	Service is not available
	kAvailable= 1	Service is available
Description:	The state of the service from `GetServiceState()`	

⌋

8.4.1.9 Type Alias: ServiceStateHandler

[SWS_CM_01072] Definition of API type ara::com::ServiceStateHandler ⌈

Kind:	type alias
Header file:	#include "ara/com/types.h"
Scope:	`namespace ara::com`
Symbol:	ServiceStateHandler
Syntax:	`using ServiceStateHandler = std::function<void(ServiceState)>;`
Thread Safety:	implementation defined
Description:	Functor used as input arguments for **any** of the `SetServiceStateChangeHandler()` methods: • [SWS_CM_01074] • [SWS_CM_01076]

⌋

8.4.1.10 Enumeration: SubscriptionState

[SWS_CM_00310] Definition of API enum ara::com::SubscriptionState

Upstream requirements: RS_CM_00103, RS_CM_00104, RS_CM_00106

⌈

Kind:	enumeration	
Header file:	#include "ara/com/types.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com`	
Symbol:	SubscriptionState	
Underlying type:	std::uint8_t	
Syntax:	`enum class SubscriptionState : std::uint8_t {...};`	
Values:	kSubscribed	Subscription is active
	kNotSubscribed	Subscription is not active
	kSubscriptionPending	Subscription is pending
Description:	The subscription state of an event.	

⌋

8.4.1.11 Type Alias: SubscriptionStateChangeHandler

[SWS_CM_00311] Definition of API type ara::com::SubscriptionStateChangeHandler

Upstream requirements: RS_CM_00103, RS_CM_00104, RS_CM_00106

⌈

Kind:	type alias
Header file:	#include "ara/com/types.h"
Scope:	`namespace ara::com`
Symbol:	SubscriptionStateChangeHandler
Syntax:	`using SubscriptionStateChangeHandler = std::function<void(SubscriptionState)>;`
Thread Safety:	implementation defined
Description:	Callback function invoked if `ara::com::SubscriptionState` of an event has changed.

⌋

8.4.1.12 Type Alias: TriggerReceiveHandler

[SWS_CM_00351] Definition of API type ara::com::TriggerReceiveHandler

Upstream requirements: RS_CM_00203, RS_AP_00114, RS_AP_00120

⌈

Kind:	type alias
Header file:	#include "ara/com/types.h"
Scope:	`namespace ara::com`
Symbol:	TriggerReceiveHandler
Syntax:	`using TriggerReceiveHandler = std::function<void()>;`
Thread Safety:	implementation defined
Description:	A function wrapper for the handler function that gets called in case a new trigger arrives for an event. The trigger receiver must provide the function implementation which is not required to be re-entrant. Usage of `std::function` is recommended but not mandatory.

⌋

8.5 Header: ara/com/runtime.h

[SWS_CM_11379] `ara::com Runtime Header File`: file name

Upstream requirements: RS_CM_00001, RS_AP_00114, RS_AP_00116

⌈

Kind:	Header File
Syntax:	`ara/com/runtime.h`
Description:	File name for `ara::com Runtime Header File` The `ara::com Runtime Header File` provides general non-service-related API functions. definitions

⌋

8.5.1 Namespaces

8.5.1.1 ara::com::runtime

[SWS_CM_11377] Definition of Namespace ara::com::runtime

Upstream requirements: RS_CM_00002, RS_AP_00114

⌈

Kind:	namespace
Header file:	#include "ara/com/runtime.h"
Scope:	`namespace ara::com`
Syntax:	`namespace runtime`
Description:	Inner namespace for non-skeleton/proxy definitions

⌋

8.5.2 Non-Member Functions

8.5.2.1 Other

8.5.2.1.1 ResolveInstanceIDs

[SWS_CM_00118] Definition of API function ara::com::runtime::ResolveInstanceIDs

Upstream requirements: RS_CM_00200, RS_AP_00114, RS_AP_00115, RS_AP_00120, RS_AP_00121, RS_AP_00119, RS_AP_00127, RS_AP_00137

⌈

Kind:	function	
Header file:	#include "ara/com/runtime.h"	
Scope:	`namespace ara::com::runtime`	
Syntax:	`ara::core::Result< ara::com::InstanceIdentifierContainer > ResolveInstanceIDs (ara::core::InstanceSpecifier metaModelIdentifier) noexcept;`	
Parameters (in):	metaModelIdentifier	The `ara::core::InstanceSpecifier` to be translated.
Return value:	ara::core::Result< ara::com::InstanceIdentifierContainer >	An `ara::com::InstanceIdentifierContainer` if successful, otherwise an error code indicating the error
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComErrc::kInstanceIDNotResolvable	rollback_semantics
		The `ara::com::InstanceSpecifier` is valid but could not be resolved/mapped to an existing `ara::com::InstanceIdentifier`
Description:	Translates an `ara::core::InstanceSpecifier` to a `ara::com::InstanceIdentifierContainer`. The length/size of the `ara::com::InstanceIdentifierContainer` is relative to the number of matches.	
See also:	[SWS_CORE_08001]	

⌋

8.6 Header: ara/com/service/find_service_handle.h

8.6.1 Struct: FindServiceHandle

[SWS_CM_00303] Definition of API class ara::com::FindServiceHandle

Upstream requirements: RS_CM_00102

⌈

Kind:	struct
Header file:	#include "ara/com/service/find_service_handle.h"
Forwarding header file:	#include "ara/com/com_fwd.h"
Scope:	`namespace ara::com`
Symbol:	FindServiceHandle
Syntax:	`struct FindServiceHandle {...};`
Description:	The exact definition of `ara::com::FindServiceHandle` is Communication Management implementation specific. `ara::com::FindServiceHandle` satisfies the `[equalitycomparable]` and `[lessthancomparable]` requirements as per [31] to allow storing and managing `ara::com::FindServiceHandle`s by the application.

⌋

8.6.1.1 Public Member Functions

8.6.1.1.1 Special Member Functions

8.6.1.1.1.1 Default Constructor

[SWS_CM_00353] Definition of API function ara::com::FindServiceHandle::FindServiceHandle

Upstream requirements: RS_AP_00146

⌈

Kind:	function
Header file:	#include "ara/com/service/find_service_handle.h"
Scope:	`struct ara::com::FindServiceHandle`
Syntax:	`FindServiceHandle ()=delete;`
Description:	Default constructor deletion

⌋

8.6.1.1.1.2 Copy Assignment Operator

[SWS_CM_11528] Definition of API function ara::com::FindServiceHandle::operator= ⌈

Kind:	function	
Header file:	#include "ara/com/service/find_service_handle.h"	
Scope:	`struct ara::com::FindServiceHandle`	
Syntax:	`FindServiceHandle & operator= (const FindServiceHandle &other);`	
Parameters (in):	other	Other `ara::com::FindServiceHandle` to copy
Return value:	FindServiceHandle &	New `ara::com::FindServiceHandle`
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Copy assignment constructor	

⌋

8.6.1.1.2 Member Functions

8.6.1.1.2.1 operator<

[SWS_CM_11527] Definition of API function ara::com::FindServiceHandle::operator< ⌈

Kind:	function	
Header file:	#include "ara/com/service/find_service_handle.h"	
Scope:	`struct ara::com::FindServiceHandle`	
Syntax:	`bool operator< (const FindServiceHandle &other) const;`	
Parameters (in):	other	Other `ara::com::FindServiceHandle` to compare
Return value:	bool	`TRUE` if `other` is less than `*this`, `FALSE` otherwise.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	LessThan operator as per `[lessthancomparable]` in [31]	

⌋

8.6.1.1.2.2 operator==

[SWS_CM_11526] Definition of API function ara::com::FindServiceHandle::operator== ⌈

Kind:	function	
Header file:	#include "ara/com/service/find_service_handle.h"	
Scope:	`struct ara::com::FindServiceHandle`	
Syntax:	`bool operator== (const FindServiceHandle &other) const;`	
Parameters (in):	other	Other `ara::com::FindServiceHandle` to compare
Return value:	bool	`TRUE` if `other` is equal to `*this`, `FALSE` otherwise.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Equality operator as per `[equalitycomparable]` in [31]	

⌋

8.7 Header: ara/com/service/instance_identifier.h

8.7.1 Class: InstanceIdentifier

[SWS_CM_00302] Definition of API class ara::com::InstanceIdentifier

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	class
Header file:	#include "ara/com/service/instance_identifier.h"
Forwarding header file:	#include "ara/com/com_fwd.h"
Scope:	`namespace ara::com`
Symbol:	InstanceIdentifier
Syntax:	`class InstanceIdentifier {...};`
Description:	Unique identifier of a specific `AdaptivePlatformServiceInstance`, to distinguish different instances of exactly the same service. `ara::com::InstanceIdentifier` satisfies the `[equalitycomparable]`, `[lessthancomparable]` and `[copyassignable]` requirements as per [31] to allow for logging of `ara::com::InstanceIdentifier`s as well as storing and managing `ara::com::InstanceIdentifier`s by the application (see operators). `ara::com::InstanceIdentifier` does not contain a fully qualified name, which would also have service type information.

⌋

8.7.1.1 Public Member Functions

8.7.1.1.1 Special Member Functions

8.7.1.1.1.1 Copy Constructor

[SWS_CM_00056] Definition of API function ara::com::InstanceIdentifier::InstanceIdentifier&

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`explicit InstanceIdentifier (const InstanceIdentifier &other) noexcept;`	
Parameters (in):	other	Other `ara::com::InstanceIdentifier` object to copy
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Copy Constructor	

⌋

8.7.1.1.1.2 Copy Assignment Operator

[SWS_CM_11525] Definition of API function ara::com::InstanceIdentifier::operator=

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`InstanceIdentifier & operator= (const InstanceIdentifier &other) noexcept;`	
Parameters (in):	other	Other `ara::com::InstanceIdentifier` to copy
Return value:	InstanceIdentifier &	New `ara::com::InstanceIdentifier`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Copy assignment constructor	

⌋

8.7.1.1.1.3 Move Assignment Operator

[SWS_CM_00054] Definition of API function ara::com::InstanceIdentifier::operator&=

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`InstanceIdentifier & operator= (InstanceIdentifier &&other) noexcept;`	
Parameters (in):	other	Other `ara::com::InstanceIdentifier` to move
Return value:	InstanceIdentifier &	New `ara::com::InstanceIdentifier`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move assignment constructor	

⌋

8.7.1.1.1.4 Destructor

[SWS_CM_00055] Definition of API function ara::com::InstanceIdentifier::~InstanceIdentifier

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function
Header file:	#include "ara/com/service/instance_identifier.h"
Scope:	`class ara::com::InstanceIdentifier`
Syntax:	`~InstanceIdentifier () noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor

⌋

8.7.1.1.2 Constructors

8.7.1.1.2.1 InstanceIdentifier

[SWS_CM_11521] Definition of API function ara::com::InstanceIdentifier::InstanceIdentifier

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`explicit InstanceIdentifier (ara::core::StringView serializedFormat);`	
Parameters (in):	serializedFormat	String form of an `ara::com::InstanceIdentifier`
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidInstanceIdentifierStringViolation`	Given `ara::com::InstanceIdentifier` string is corrupted, non-compliant or duplicated
Description:	Default Constructor	

⌋

8.7.1.1.2.2 InstanceIdentifier

[SWS_CM_00053] Definition of API function ara::com::InstanceIdentifier::InstanceIdentifier&&

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`explicit InstanceIdentifier (const InstanceIdentifier &&other) noexcept;`	
Parameters (in):	other	Other `ara::com::InstanceIdentifier` object to move
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move Constructor	

⌋

8.7.1.1.3 Member Functions

8.7.1.1.3.1 Create

[SWS_CM_11520] Definition of API function ara::com::InstanceIdentifier::Create

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`static InstanceIdentifier Create (ara::core::StringView serializedFormat) noexcept;`	
Parameters (in):	serializedFormat	String form of an `ara::com::InstanceIdentifier`
Return value:	InstanceIdentifier	An `ara::com::InstanceIdentifier`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Violations:	`InvalidInstanceIdentifierStringViolation`	Given `ara::com::InstanceIdentifier` string is corrupted, non-compliant or duplicated
Description:	Exception-less construction of a `ara::com::InstanceIdentifier`	

⌋

8.7.1.1.3.2 operator<

[SWS_CM_11524] Definition of API function ara::com::InstanceIdentifier::operator<

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`bool operator< (const InstanceIdentifier &other) const noexcept;`	
Parameters (in):	other	Other `ara::com::InstanceIdentifier` to compare
Return value:	bool	`TRUE` if `other` is less than `*this`, `FALSE` otherwise.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	LessThan operator as per `[lessthancomparable]` in [31]	

⌋

8.7.1.1.3.3 operator==

[SWS_CM_11523] Definition of API function ara::com::InstanceIdentifier::operator==

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`bool operator== (const InstanceIdentifier &other) const noexcept;`	
Parameters (in):	other	Other `ara::com::InstanceIdentifier` to compare
Return value:	bool	`TRUE` if `other` is equal to `*this`, `FALSE` otherwise.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Equality operator as per `[equalitycomparable]` in [31]	

⌋

8.7.1.1.3.4 toString

[SWS_CM_11522] Definition of API function ara::com::InstanceIdentifier::toString

Upstream requirements: RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00127

⌈

Kind:	function	
Header file:	#include "ara/com/service/instance_identifier.h"	
Scope:	`class ara::com::InstanceIdentifier`	
Syntax:	`ara::core::StringView toString () const noexcept;`	
Return value:	ara::core::StringView	String representation of a `ara::com::InstanceIdentifier`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Stringification method for `ara::com::InstanceIdentifier`	

⌋

8.8 Header: ara/com/service/sample_ptr.h

8.8.1 Class: SamplePtr

[SWS_CM_00306] Definition of API class ara::com::SamplePtr

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	class	
Header file:	#include "ara/com/service/sample_ptr.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com`	
Symbol:	SamplePtr	
Syntax:	`template <typename T>` `class SamplePtr {...};`	
Template param:	typename T	A managed object (event sample)
Description:	Emulates a `std::unique_ptr` to an event sample. The `ara::com::SamplePtr` behaves as a `std::unique_ptr` as long as the `event`/ `field` is subscribed to, or the `Proxy` it belongs to is not destroyed. The precondition defined in [SWS_CM_00085] and [SWS_CM_00087] must be fulfilled. Otherwise it is considered a violation.	

⌋

8.8.1.1 Public Member Functions

8.8.1.1.1 Special Member Functions

8.8.1.1.1.1 Copy Constructor

[SWS_CM_11536] Definition of API function ara::com::SamplePtr::SamplePtr

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/com/service/sample_ptr.h"
Scope:	`class ara::com::SamplePtr`
Syntax:	`SamplePtr (const SamplePtr &)=delete;`
Description:	Copy constructor deletion

⌋

8.8.1.1.1.2 Default Constructor

[SWS_CM_11534] Definition of API function ara::com::SamplePtr::SamplePtr

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/com/service/sample_ptr.h"
Scope:	`class ara::com::SamplePtr`
Syntax:	`constexpr SamplePtr () noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Default constructor

⌋

8.8.1.1.1.3 Move Constructor

[SWS_CM_11537] Definition of API function ara::com::SamplePtr::SamplePtr

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`SamplePtr (SamplePtr &&other) noexcept;`	
Parameters (in):	other	Other `ara::com::SamplePtr` to move
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move constructor	

⌋

8.8.1.1.1.4 Move Assignment Operator

[SWS_CM_11540] Definition of API function ara::com::SamplePtr::operator=

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`SamplePtr & operator= (SamplePtr &&other) noexcept;`	
Parameters (in):	other	Other `ara::com::SamplePtr` to move
Return value:	SamplePtr &	New `ara::com::SamplePtr`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Move assignment operator.	

⌋

8.8.1.1.1.5 Copy Assignment Operator

[SWS_CM_11538] Definition of API function ara::com::SamplePtr::operator=

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/com/service/sample_ptr.h"
Scope:	`class ara::com::SamplePtr`
Syntax:	`SamplePtr & operator= (const SamplePtr &)=delete;`
Description:	Copy assignment constructor deletion

⌋

8.8.1.1.1.6 Destructor

[SWS_CM_11547] Definition of API function ara::com::SamplePtr::~SamplePtr

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function
Header file:	#include "ara/com/service/sample_ptr.h"
Scope:	`class ara::com::SamplePtr`
Syntax:	`~SamplePtr () noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Destructor

⌋

8.8.1.1.2 Constructors

8.8.1.1.2.1 SamplePtr

[SWS_CM_11535] Definition of API function ara::com::SamplePtr::SamplePtr

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`constexpr SamplePtr (std::nullptr_t other) noexcept;`	
Parameters (in):	other	A `std::nullptr_t` to copy
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Semantically equivalent to [SWS_CM_11534]	

⌋

8.8.1.1.3 Member Functions

8.8.1.1.3.1 Get

[SWS_CM_11546] Definition of API function ara::com::SamplePtr::Get

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`T * Get () const noexcept;`	
Return value:	T *	A pointer to the managed object
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns a pointer to the managed object.	

⌋

8.8.1.1.3.2 GetProfileCheckStatus

[SWS_CM_90420] Definition of API function ara::com::SamplePtr::GetProfileCheckStatus

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`ara::com::e2e::ProfileCheckStatus GetProfileCheckStatus () const noexcept;`	
Return value:	ara::com::e2e::ProfileCheckStatus	The `ara::com::e2e::ProfileCheckStatus`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns the E2E protection check result	

⌋

8.8.1.1.3.3 Reset

[SWS_CM_11545] Definition of API function ara::com::SamplePtr::Reset

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`void Reset (std::nullptr_t other) noexcept;`	
Parameters (in):	other	A replacing `std::nullptr_t`
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Replaces the managed object.	

⌋

8.8.1.1.3.4 Swap

[SWS_CM_11544] Definition of API function ara::com::SamplePtr::Swap

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`void Swap (SamplePtr &other) noexcept;`	
Parameters (in):	other	Another `ara::com::SamplePtr` to swap the managed object with
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Swaps the managed object.	

⌋

8.8.1.1.3.5 operator bool

[SWS_CM_11543] Definition of API function ara::com::SamplePtr::operator bool

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`explicit operator bool () const noexcept;`	
Return value:	bool	`TRUE` if the stored pointer is null, `FALSE` otherwise.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Checks if the stored pointer is null	

⌋

8.8.1.1.3.6 operator*

[SWS_CM_11541] Definition of API function ara::com::SamplePtr::operator*

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`T & operator* () const noexcept;`	
Return value:	T &	The object owned by `*this`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Dereferences the stored pointer.	

⌋

8.8.1.1.3.7 operator->

[SWS_CM_11542] Definition of API function ara::com::SamplePtr::operator->

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`T * operator-> () const noexcept;`	
Return value:	T *	A pointer to the object owned by `*this`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Dereferences the stored pointer.	

⌋

8.8.1.1.3.8 operator=

[SWS_CM_11539] Definition of API function ara::com::SamplePtr::operator=

Upstream requirements: RS_CM_00202, RS_CM_00203, RS_AP_00114, RS_AP_00122, RS_AP_00132, RS_AP_00135, RS_AP_00145

⌈

Kind:	function	
Header file:	#include "ara/com/service/sample_ptr.h"	
Scope:	`class ara::com::SamplePtr`	
Syntax:	`SamplePtr & operator= (std::nullptr_t other) noexcept;`	
Parameters (in):	other	A `std::nullptr_t` to copy
Return value:	SamplePtr &	New `ara::com::SamplePtr`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Copy assignment of nullptr_t.	

⌋

8.9 Header: ara/com/service/service_identifier.h

8.9.1 Class: ServiceIdentifierType

[SWS_CM_11510] Definition of API class ara::com::ServiceIdentifierType

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	class
Header file:	#include "ara/com/service/service_identifier.h"
Forwarding header file:	#include "ara/com/com_fwd.h"
Scope:	`namespace ara::com`
Symbol:	ServiceIdentifierType
Syntax:	`class ServiceIdentifierType {...};`
Description:	Specifies a Service Identifier The exact type may be overridden by the Communication Management software provider. but shall at least satisfy the `[equalitycomparable]` ([SWS_CM_11511]), `[lessthancomparable]` ([SWS_CM_11512]) and `[copyassignable]` ([SWS_CM_11513]) requirements as per [31] to allow for logging, storing and managing `ara::com::ServiceIdentifierType`s by the application.

⌋

8.9.1.1 Public Member Functions

8.9.1.1.1 Special Member Functions

8.9.1.1.1.1 Copy Assignment Operator

[SWS_CM_11513] Definition of API function ara::com::ServiceIdentifierType::operator=

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	function	
Header file:	#include "ara/com/service/service_identifier.h"	
Scope:	`class ara::com::ServiceIdentifierType`	
Syntax:	`ServiceIdentifierType & operator= (const ServiceIdentifierType &other);`	
Parameters (in):	other	`ara::com::ServiceIdentifierType` to copy
Return value:	ServiceIdentifierType &	New `ara::com::ServiceIdentifierType`
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Copy assignment constructor	

⌋

8.9.1.1.2 Member Functions

8.9.1.1.2.1 operator<

[SWS_CM_11512] Definition of API function ara::com::ServiceIdentifierType::operator<

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	function	
Header file:	#include "ara/com/service/service_identifier.h"	
Scope:	`class ara::com::ServiceIdentifierType`	
Syntax:	`bool operator< (const ServiceIdentifierType &other) const;`	
Parameters (in):	other	Other `ara::com::ServiceIdentifierType` to compare
Return value:	bool	`TRUE` if `other` is less than `*this`, `FALSE` otherwise.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	LessThan operator as per `[lessthancomparable]` in [31] LessThan operator as per `[lessthancomparable]` in [31]	

⌋

8.9.1.1.2.2 operator==

[SWS_CM_11511] Definition of API function ara::com::ServiceIdentifierType::operator==

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	function	
Header file:	#include "ara/com/service/service_identifier.h"	
Scope:	`class ara::com::ServiceIdentifierType`	
Syntax:	`bool operator== (const ServiceIdentifierType &other) const;`	
Parameters (in):	other	`ara::com::ServiceIdentifierType` to compare
Return value:	bool	`TRUE` if `other` is equal to `*this`, `FALSE` otherwise.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Equality operator as per `[equalitycomparable]` in [31]	

⌋

8.9.1.1.2.3 toString

[SWS_CM_11514] Definition of API function ara::com::ServiceIdentifierType::toString

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	function	
Header file:	#include "ara/com/service/service_identifier.h"	
Scope:	`class ara::com::ServiceIdentifierType`	
Syntax:	`ara::core::StringView toString () const;`	
Return value:	ara::core::StringView	String representation of a `ara::com::ServiceIdentifierType`
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Stringification method for `ara::com::ServiceIdentifierType`	

⌋

8.10 Header: ara/com/service/service_version.h

8.10.1 Class: ServiceVersionType

[SWS_CM_11515] Definition of API class ara::com::ServiceVersionType

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	class
Header file:	#include "ara/com/service/service_version.h"
Forwarding header file:	#include "ara/com/com_fwd.h"
Scope:	`namespace ara::com`
Symbol:	ServiceVersionType
Syntax:	`class ServiceVersionType {...};`
Description:	Specifies a Service Version. The exact type may be overridden by the Communication Management software provider. but shall at least satisfy the `[equalitycomparable]` ([SWS_CM_11516]), `[lessthancomparable]` ([SWS_CM_11517]) and `[copyassignable]` ([SWS_CM_11518]) requirements as per [31] to allow for logging, storing and managing `ara::com::ServiceVersionType`s by the application.

⌋

8.10.1.1 Public Member Functions

8.10.1.1.1 Special Member Functions

8.10.1.1.1.1 Copy Assignment Operator

[SWS_CM_11518] Definition of API function ara::com::ServiceVersionType::operator=

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	function	
Header file:	#include "ara/com/service/service_version.h"	
Scope:	`class ara::com::ServiceVersionType`	
Syntax:	`ServiceVersionType & operator= (const ServiceVersionType &other);`	
Parameters (in):	other	`ara::com::ServiceVersionType` to copy
Return value:	ServiceVersionType &	New `ara::com::ServiceVersionType`
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Copy assignment constructor	

⌋

8.10.1.2 Member Functions

8.10.1.2.1 ToString

[SWS_CM_11519] Definition of API function ara::com::ServiceVersionType::ToString

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	function	
Header file:	#include "ara/com/service/service_version.h"	
Scope:	`class ara::com::ServiceVersionType`	
Syntax:	`ara::core::StringView ToString () const;`	
Return value:	ara::core::StringView	String representation of a `ara::com::ServiceVersionType`
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Stringification method for `ara::com::ServiceVersionType`	

⌋

8.10.1.1.2.2 operator<

[SWS_CM_11517] Definition of API function ara::com::ServiceVersionType::operator<

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	function	
Header file:	#include "ara/com/service/service_version.h"	
Scope:	`class ara::com::ServiceVersionType`	
Syntax:	`bool operator< (const ServiceVersionType &other) const;`	
Parameters (in):	other	`ara::com::ServiceVersionType` to compare
Return value:	bool	`TRUE` if `other` is less than `*this`, `FALSE` otherwise.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	LessThan operator as per `[lessthancomparable]` in [31]	

⌋

8.10.1.1.2.3 operator==

[SWS_CM_11516] Definition of API function ara::com::ServiceVersionType::operator==

Upstream requirements: RS_CM_00200, RS_CM_00500

⌈

Kind:	function	
Header file:	#include "ara/com/service/service_version.h"	
Scope:	`class ara::com::ServiceVersionType`	
Syntax:	`bool operator== (const ServiceVersionType &other) const;`	
Parameters (in):	other	`ara::com::ServiceVersionType` to compare
Return value:	bool	`TRUE` if `other` is equal to `*this`, `FALSE` otherwise.
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Equality operator as per `[equalitycomparable]` in [31]	

⌋

8.11 Header: ara/com/com_error_domain.h

8.11.1 Namespaces

8.11.1.1 ara::com

[SWS_CM_01018] `ara::com Types Header File`: namespace

Upstream requirements: RS_AP_00114, RS_CM_00002

⌈

Kind:	namespace
Header file:	#include "ara/com/com_error_domain.h"
Scope:	`namespace ara`
Syntax:	`namespace com`
Description:	Namespace for `ara::com Types Header File` definitions

⌋

8.11.2 Non-Member Types

8.11.2.1 Enumeration: ComErrc

[SWS_CM_10432] Definition of API enum ara::com::ComErrc

Upstream requirements: RS_AP_00130, RS_AP_00122, RS_AP_00127

⌈

Kind:	enumeration	
Header file:	#include "ara/com/com_error_domain.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com`	
Symbol:	ComErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class ComErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kServiceNotAvailable= 1	Service is not available after being previously offered via `OfferService()`
	kMaxSamples Exceeded= 2	Application holds more `ara::com::SamplePtr`s than commited in `Subscribe()` Samples were possibly skipped.
	kNetworkBindingFailure= 3	The network binding reported a recoverable communications error or a secure communication failure
	kFieldValueNot Initialized= 6	Field value has yet not been initialized via `Update()` (see [SWS_CM_00128]

▽

△

	kFieldSetHandlerNot Set= 7	Field SetHandler has not been registered via `RegisterSetHandler()`
	kServiceNotOffered= 11	Service has not yet been offered via `OfferService()`
	kInstanceIDNot Resolvable= 15	The `ara::com::InstanceSpecifier` is valid but could not be resolved/mapped to an existing `ara::com::InstanceIdentifier`
	kMaxSampleCountNot Realizable= 16	Provided `maxSampleCount` for event re-subscription does not match the `maxSampleCount` for the current subscription
	kUnknownApplication Error= 22	A remote service returned an unconfigured application error.
Description:	Defines the error codes for the `ara::com::ComErrorDomain`	

⌋

8.11.3 Non-Member Functions

8.11.3.1 Other

8.11.3.1.1 GetComErrorDomain

[SWS_CM_11334] Definition of API function ara::com::GetComErrorDomain

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/com_error_domain.h"	
Scope:	`namespace ara::com`	
Syntax:	`constexpr const ara::core::ErrorDomain & GetComErrorDomain () noexcept;`	
Return value:	const ara::core::Error Domain &	Reference to the `ara::com::ComErrorDomain` object
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns a reference to the `ara::com::ComErrorDomain` object	

⌋

8.11.3.1.2 MakeErrorCode

[SWS_CM_11335] Definition of API function ara::com::MakeErrorCode

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/com_error_domain.h"	
Scope:	`namespace ara::com`	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (ara::com::ComErrc 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 `ara::core::ErrorCode` object.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Creates an instance of `ara::core::ErrorCode`	

⌋

8.11.4 Class: ComErrorDomain

[SWS_CM_11329] Definition of API class ara::com::ComErrorDomain

Upstream requirements: RS_AP_00130, RS_AP_00122, RS_AP_00127

⌈

Kind:	class
Header file:	#include "ara/com/com_error_domain.h"
Forwarding header file:	#include "ara/com/com_fwd.h"
Scope:	`namespace ara::com`
Symbol:	ComErrorDomain
Base class:	ara::core::ErrorDomain
Syntax:	`class ComErrorDomain final : public ara::core::ErrorDomain {...};`
Unique ID:	As per `ara::com::ComErrorDomain` in [SWS_CORE_90023]
Description:	Defines a class representing the Communication error domain.

⌋

8.11.4.1 Public Member Types

8.11.4.1.1 Type Alias: Errc

[SWS_CM_11336] Definition of API type ara::com::ComErrorDomain::Errc

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	type alias
Header file:	#include "ara/com/com_error_domain.h"
Scope:	`class ara::com::ComErrorDomain`
Symbol:	Errc
Syntax:	`using Errc = ComErrc;`
Description:	Alias for the error code value enumeration

⌋

8.11.4.1.2 Type Alias: Exception

[SWS_CM_11337] Definition of API type ara::com::ComErrorDomain::Exception

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	type alias
Header file:	#include "ara/com/com_error_domain.h"
Scope:	`class ara::com::ComErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = ComException;`
Description:	Alias for the exception base class

⌋

8.11.4.2 Public Member Functions

8.11.4.2.1 Special Member Functions

8.11.4.2.1.1 Default Constructor

[SWS_CM_11330] Definition of API function ara::com::ComErrorDomain::ComErrorDomain

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	function
Header file:	#include "ara/com/com_error_domain.h"
Scope:	`class ara::com::ComErrorDomain`
Syntax:	`ComErrorDomain ()=delete;`
Description:	Constructs a new `ara::com::ComErrorDomain` object

⌋

8.11.4.2 Member Functions

8.11.4.2.1 Message

[SWS_CM_11332] Definition of API function ara::com::ComErrorDomain::Message

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/com_error_domain.h"	
Scope:	`class ara::com::ComErrorDomain`	
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 `errorCode`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns the message associated with `errorCode`	

⌋

8.11.4.2.2.2 Name

[SWS_CM_11331] Definition of API function ara::com::ComErrorDomain::Name

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/com_error_domain.h"	
Scope:	`class ara::com::ComErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Com"
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns a string constant associated with the `ara::com::ComErrorDomain`	

⌋

8.11.4.2.2.3 ThrowAsException

[SWS_CM_11333] Definition of API function ara::com::ComErrorDomain::ThrowAsException

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/com/com_error_domain.h"	
Scope:	`class ara::com::ComErrorDomain`	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Creates a new instance of `ara::com::ComException` from `errorCode` and throws it. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.11.5 Class: ComException

[SWS_CM_11327] Definition of API class ara::com::ComException

Upstream requirements: RS_AP_00130, RS_AP_00122, RS_AP_00127

⌈

Kind:	class
Header file:	#include "ara/com/com_error_domain.h"
Forwarding header file:	#include "ara/com/com_fwd.h"
Scope:	`namespace ara::com`
Symbol:	ComException
Base class:	ara::core::Exception
Syntax:	`class ComException final : public ara::core::Exception {...};`
Description:	Defines a class for exceptions to be thrown by the API.

⌋

8.11.5.1 Public Member Functions

8.11.5.1.1 Constructors

8.11.5.1.1.1 ComException

[SWS_CM_11328] Definition of API function ara::com::ComException::ComException

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/com_error_domain.h"	
Scope:	`class ara::com::ComException`	
Syntax:	`explicit ComException (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Constructs a new `ara::com::ComException` containing an `ara::core::ErrorCode`	

⌋

8.12 Header: ara/com/e2e/profile_check_status.h

8.12.1 Non-Member Types

8.12.1.1 Enumeration: ProfileCheckStatus

[SWS_CM_90421] Definition of API enum ara::com::e2e::ProfileCheckStatus

Upstream requirements: RS_E2E_08534, RS_AP_00114, RS_AP_00115, RS_AP_00119

⌈

Kind:	enumeration	
Header file:	#include "ara/com/e2e/profile_check_status.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com::e2e`	
Symbol:	ProfileCheckStatus	
Underlying type:	std::uint8_t	
Syntax:	`enum class ProfileCheckStatus : std::uint8_t {...};`	
Values:	kOk	The checks of the sample in this cycle were successful (including counter check)
	kRepeated	Sample has a repeated counter
	kWrongSequence	The checks of the sample in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta.
	kError	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong Data ID).
	kCheckDisabled	No E2E check status available. Return value of function GetProfile CheckStatus if `EndToEndTransformationComSpecProps.disableEndToEndCheck` is set to `TRUE`
Description:	The result of the check of a single `ara::com::SamplePtr`	
See also:	[PRS_E2E_00322], [PRS_E2E_00677]	

⌋

8.13 Header: ara/com/e2e/sm_state.h

8.13.1 Non-Member Types

8.13.1.1 Enumeration: SMState

[SWS_CM_90422] Definition of API enum ara::com::e2e::SMState

Status: DRAFT
Upstream requirements: RS_E2E_08534, RS_AP_00114, RS_AP_00115, RS_AP_00119

⌈

Kind:	enumeration	
Header file:	#include "ara/com/e2e/sm_state.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com::e2e`	
Symbol:	SMState	
Underlying type:	std::uint8_t	
Syntax:	`enum class SMState : std::uint8_t {...};`	
Values:	kValid	Communication of the samples of this event is functioning properly according to E2E checks; sample(s) can be used
	kNoData	No data have been received from the publisher at all
	kInit	Not enough data where the E2E check yielded OK from the publisher is available since the initialization; sample(s) cannot be used
	kInvalid	Too little data (where the E2E check yielded OK), or too much data (where the e2e check yielded ERROR) were received within the E2E time window - communication of the sample of this event not functioning properly; sample(s) cannot be used
	kStateMDisabled	No E2E state machine available. Return value of function Get SMState if `EndToEndTransformationComSpec Props.disableEndToEndCheck` == `TRUE`
	kIncompatibleQoS	Samples can't be received because configured QoS policies at each end of communication don't match. (Only applicable to certain network bindings.)
Description:	The state of the E2E supervision after the most recent check of a received sample of the event. If `ara::com::e2e::SMState` == `kValid`, and the `ara::com::SamplePtr::GetProfileCheckStatus` did not result in an Error then, the last checked sample can be used.	
See also:	[PRS_E2E_00322], [PRS_E2E_00678]	

⌋

8.14 Header: ara/com/e2e/e2e_error_domain.h

8.14.1 Non-Member Types

8.14.1.1 Enumeration: ComE2EErrc

[SWS_CM_10474] Definition of API enum ara::com::e2e::ComE2EErrc

Upstream requirements: RS_AP_00130

⌈

Kind:	enumeration	
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com::e2e`	
Symbol:	ComE2EErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class ComE2EErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kRepeated= 1	Data has a repeated counter
	kWrongSequence= 2	The checks of the Data in this cycle were successful, with the exception of counter jump, which changed more than the allowed delta
	kError= 3	Error not related to counters occurred (e.g. wrong crc, wrong length, wrong Data ID) or the return of the check function was not OK
	kNoNewData= 5	No new data is available
	kUnspecifiedE2EError= 6	A remote service returned an unspecified E2E error.
Description:	Defines the error codes for the `ara::com::e2e::ComE2EErrc`	

⌋

8.14.2 Non-Member Functions

8.14.2.1 Other

8.14.2.1.1 GetComE2EErrorDomain

[SWS_CM_12510] Definition of API function ara::com::e2e::GetComE2EErrorDomain

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Scope:	`namespace ara::com::e2e`	
Syntax:	`constexpr const ara::core::ErrorDomain & GetComE2EErrorDomain () noexcept;`	
Return value:	const ara::core::ErrorDomain &	Reference to the `ara::com::e2e::ComE2EErrorDomain` object
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns a reference to the `ara::com::e2e::ComE2EErrorDomain` object	

⌋

8.14.2.1.2 MakeErrorCode

[SWS_CM_12511] Definition of API function ara::com::e2e::MakeErrorCode

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Scope:	`namespace ara::com::e2e`	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (ara::com::e2e::ComE2EErrc 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 `ara::core::ErrorCode` object.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Creates an instance of `ara::core::ErrorCode`	

⌋

8.14.3 Class: ComE2EErrorDomain

[SWS_CM_12503] Definition of API class ara::com::e2e::ComE2EErrorDomain

Upstream requirements: RS_AP_00122, RS_AP_00127, RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/com/e2e/e2e_error_domain.h"
Forwarding header file:	#include "ara/com/com_fwd.h"
Scope:	`namespace ara::com::e2e`
Symbol:	ComE2EErrorDomain
Base class:	ara::core::ErrorDomain
Syntax:	`class ComE2EErrorDomain final : public ara::core::ErrorDomain {...};`
Unique ID:	As per `ara::com::ComE2EErrorDomain` in [SWS_CORE_90023]
Description:	Defines a class representing the E2E error domain

⌋

8.14.3.1 Public Member Types

8.14.3.1.1 Type Alias: Errc

[SWS_CM_12504] Definition of API type ara::com::e2e::ComE2EErrorDomain::Errc

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	type alias
Header file:	#include "ara/com/e2e/e2e_error_domain.h"
Scope:	`class ara::com::e2e::ComE2EErrorDomain`
Symbol:	Errc
Syntax:	`using Errc = ComE2EErrc;`
Description:	Alias for the error code value enumeration

⌋

8.14.3.1.2 Type Alias: Exception

[SWS_CM_12505] Definition of API type ara::com::e2e::ComE2EErrorDomain::Exception

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	type alias
Header file:	#include "ara/com/e2e/e2e_error_domain.h"
Scope:	`class ara::com::e2e::ComE2EErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = ComE2EException;`
Description:	Alias for the exception base class

⌋

8.14.3.2 Public Member Functions

8.14.3.2.1 Special Member Functions

8.14.3.2.1.1 Default Constructor

[SWS_CM_12506] Definition of API function ara::com::e2e::ComE2EErrorDomain::ComE2EErrorDomain

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	function
Header file:	#include "ara/com/e2e/e2e_error_domain.h"
Scope:	`class ara::com::e2e::ComE2EErrorDomain`
Syntax:	`constexpr ComE2EErrorDomain () noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Constructs a new `ara::com::e2e::ComE2EErrorDomain` object

⌋

8.14.3.2.2 Member Functions

8.14.3.2.2.1 Message

[SWS_CM_12508] Definition of API function ara::com::e2e::ComE2EErrorDomain::Message

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Scope:	`class ara::com::e2e::ComE2EErrorDomain`	
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 `errorCode`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns the message associated with `errorCode`	

⌋

8.14.3.2.2.2 Name

[SWS_CM_12507] Definition of API function ara::com::e2e::ComE2EErrorDomain::Name

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Scope:	`class ara::com::e2e::ComE2EErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"ComE2E"
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns a string constant associated with the `ara::com::e2e::ComE2EErrorDomain`	

⌋

8.14.3.2.2.3 ThrowAsException

[SWS_CM_12509] Definition of API function ara::com::e2e::ComE2EErrorDomain::ThrowAsException

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130

⌈

Kind:	function	
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Scope:	`class ara::com::e2e::ComE2EErrorDomain`	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Creates a new instance of `ara::com::e2e::ComE2EException` from `errorCode` and throws it. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.14.4 Class: ComE2EException

[SWS_CM_12501] Definition of API class ara::com::e2e::ComE2EException

Upstream requirements: RS_AP_00122, RS_AP_00127, RS_AP_00130

⌈

Kind:	class
Header file:	#include "ara/com/e2e/e2e_error_domain.h"
Forwarding header file:	#include "ara/com/com_fwd.h"
Scope:	`namespace ara::com::e2e`
Symbol:	ComE2EException
Base class:	ara::core::Exception
Syntax:	`class ComE2EException final : public ara::core::Exception {...};`
Description:	Defines a class for exceptions to be thrown by the E2E APIs

⌋

8.14.4.1 Public Member Functions

8.14.4.1.1 Constructors

8.14.4.1.1.1 ComE2EException

[SWS_CM_12502] Definition of API function ara::com::e2e::ComE2EException::ComE2EException

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Scope:	`class ara::com::e2e::ComE2EException`	
Syntax:	`explicit ComE2EException (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Constructs a new `ara::com::e2e::ComE2EException` object containing an error code	

⌋

8.15 Header: ara/com/e2e/transport_fault_condition.h

8.15.1 Non-Member Types

8.15.1.1 Enumeration: TransportFaultCondition

[SWS_CM_00091] Definition of API enum ara::com::e2e::TransportFaultCondition ⌈

Kind:	enumeration	
Header file:	#include "ara/com/e2e/transport_fault_condition.h"	
Forwarding header file:	#include "ara/com/com_fwd.h"	
Scope:	`namespace ara::com::e2e`	
Symbol:	TransportFaultCondition	
Underlying type:	std::uint8_t	
Syntax:	`enum class TransportFaultCondition : std::uint8_t {...};`	
Values:	kSampleRejected	A sample has been rejected on layer 4 or above due to end-point resource exhaustion
	kDeadlineMissed	A sample has missed its configured deadline period

▽

△

	kSampleLost	Loss of a sample has been detected (e.g. gap in expected sequence numbering)
Description:	Defines the TransportFaultCondition which represent a single transport-related fault event	

⌋

8.15.1.2 Type Alias: TransportFaultConditionHandler

[SWS_CM_00092] Definition of API type ara::com::e2e::TransportFaultCondition Handler ⌈

Kind:	type alias
Header file:	#include "ara/com/e2e/transport_fault_condition.h"
Scope:	`namespace ara::com::e2e`
Symbol:	TransportFaultConditionHandler
Syntax:	`using TransportFaultConditionHandler = std::function<void(TransportFaultCondition, std::size_t)>;`
Thread Safety:	implementation defined
Description:	Handler type to be called when one or more (determined by the value of the `std::size_t parameter`) fault conditions are detected

⌋

9 Service Interfaces

This chapter lists the service interfaces provided or required by this functional cluster.

9.1 Implementation Data Types

[SWS_CM_11285] Definition of ImplementationDataType OverrideStatus

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Name	OverrideStatus	
Namespace	ara::com::secoc	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Override Status enum	
Range / Symbol	***Limit***	***Description***
kSecOcOverrideDropUntil Notice	0x00	Until further notice, authenticator verification is not performed, PDU is dropped, verification result is set to kSecOcNoVerification.
kSecOcOverrideDropUntilLimit	0x01	Until NumberOfMessagesToOverride is reached, authenticator verification is not performed, PDU is dropped, verification result is set to kSecOcNoVerification.
kSecOcOverrideCancel	0x02	Cancel Override of VerifyStatus.
kSecOcOverridePassUntil Notice	0x40	Until further notice, authenticator verification is performed, PDU is forwarded to the application independent of verification result, verification result is set to kSecOcVerificationFailureOverwritten in case of failed verification.
kSecOcOverrideSkipUntilLimit	0x41	Until NumberOfMessagesToOverride is reached, authenticator verification is not performed, PDU is sent to the application, verification result is set to kSecOcNoVerification.
kSecOcOverridePassUntilLimit	0x42	Until NumberOfMessagesToOverride is reached, authenticator verification is performed, PDU is sent to the application independent of verification result, verification result is set to kSecOcVerificationFailure Overwritten in case of failed verification.
kSecOcOverrideSkipUntil Notice	0x43	Until further notice, authenticator verification is not performed, PDU is sent to the application, verification result is set to kSecOcNo Verification.

⌋

[SWS_CM_11283] Definition of ImplementationDataType VerificationStatusContainer

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Name	VerificationStatusContainer
Namespace	ara::com::secoc
Kind	STRUCTURE
Sub-elements	freshnessValueID `uint16_t` verificationStatus `VerificationStatusResult` secOCDataId `uint16_t`
Derived from	-
Description	Data structure to bundle the status of a verification attempt for a specific Freshness Value and Data ID

⌋

[SWS_CM_11284] Definition of ImplementationDataType VerificationStatusResult

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Name	VerificationStatusResult	
Namespace	ara::com::secoc	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Data structure to bundle the status of a verification attempt for a specific Freshness Value and Data ID	
Range / Symbol	***Limit***	***Description***
kSecOcVerificationSuccess	0x00	Verification successful
kSecOcVerificationFailure	0x01	Verification not successful
kSecOcFreshnessFailure	0x02	Verification not successful because of wrong freshness value.
kSecOcAuthenticationBuild Failure	0x03	Verification not successful because of wrong build authentication codes
kSecOcNoVerification	0x04	Verification has been skipped and the data has been provided to the application as is.
kSecOcVerificationFailure Overwritten	0x05	Verification failed, but the PDU was passed on to the application due to the override status for this PDU.

⌋

9.2 Provided Service Interfaces

9.2.1 Provided Ports

[SWS_CM_11367] Definition of Port VerificationStatus provided by functional cluster CM

Status: DRAFT

⌈

Name	VerificationStatus		
Kind	ProvidedPort	***Interface***	`VerificationStatus`
Description	Provide services for VerificationStatus		
Variation			

⌋

[SWS_CM_11369] Definition of Port VerificationStatusConfigurationByDataId provided by functional cluster CM

Status: DRAFT

⌈

Name	VerificationStatusConfigurationByDataId		
Kind	ProvidedPort	***Interface***	`VerificationStatusConfigura-tionByDataId`
Description	Provide services for VerificationStatusConfigurationByDataId		
Variation			

⌋

[SWS_CM_11368] Definition of Port VerificationStatusConfigurationByFreshnessId provided by functional cluster CM

Status: DRAFT

⌈

Name	VerificationStatusConfigurationByFreshnessId		
Kind	ProvidedPort	***Interface***	`VerificationStatusConfigura-tionByFreshnessId`
Description	Provide services for VerificationStatusConfigurationByFreshnessId		
Variation			

⌋

9.2.2 Client -Server Interfaces

[SWS_CM_11280] Definition of ServiceInterface VerificationStatus

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Name	VerificationStatus
Namespace	ara::com::secoc
Version	1.0
Events	`VerificationStatus`

⌋

[SWS_CM_10048] Definition of Event VerificationStatus.VerificationStatus

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Event	VerificationStatus
Version	1.0
Type	`VerificationStatusContainer`
Enclosing Service Interface	`VerificationStatus`

⌋

[SWS_CM_11282] Definition of ServiceInterface VerificationStatusConfigurationByDataId

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Name	VerificationStatusConfigurationByDataId
Namespace	ara::com::secoc
Version	1.0
Methods	`VerifyStatusOverride`

⌋

[SWS_CM_10050] Definition of Method VerificationStatusConfigurationByDataId.VerifyStatusOverride

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Method	VerifyStatusOverride	
Description	This service method provides the ability to force specific behavior of SecOc: accept or drop a message with or without performing the verification of authenticator or independent of the authenticator verification result, and to force a specific result for VerificationStatusResult allowing additional fault handling in the application.	
Version	1.0	
FireAndForget	false	
Parameter	dataID	
	Description	Data ID for which the override operation shall happen
	Type	`uint16_t`
	Variation	
	Direction	IN
Parameter	overrideStatus	
	Description	The override status enum that defines whether verification is executed and whether the message is passed on, and for how long the override is active
	Type	`OverrideStatus`
	Variation	
	Direction	IN
Parameter	numberOfMessagesToOverride	
	Description	Number of sequential VerifyStatus to override when using a specific counter for authentication verification. This is only considered when OverrideStatus is equal to kSecOcOverrideDropUntilLimit, kSecOcOverrideSkipUntilLimit or kSecOcOverridePassUntilLimit.
	Type	`uint8_t`
	Variation	
	Direction	IN
Enclosing Service Interface	`VerificationStatusConfigurationByDataId`	

⌋

[SWS_CM_11281] Definition of ServiceInterface VerificationStatusConfigurationByFreshnessId

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Name	VerificationStatusConfigurationByFreshnessId
Namespace	ara::com::secoc
Version	1.0
Methods	`VerifyStatusOverride`

⌋

[SWS_CM_10049] Definition of Method VerificationStatusConfigurationByFreshnessId.VerifyStatusOverride

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Method	VerifyStatusOverride	
Description	This service method provides the ability to force specific behavior of SecOc: accept or drop a message with or without performing the verification of authenticator or independent of the authenticator verification result, and to force a specific result for VerificationStatusResult allowing additional fault handling in the application.	
Version	1.0	
FireAndForget	false	
Parameter	freshnessID	
	Description	Freshness value ID for which the override operation shall happen
	Type	`uint16_t`
	Variation	
	Direction	IN
Parameter	overrideStatus	
	Description	The override status enum that defines whether verification is executed and whether the message is passed on, and for how long the override is active
	Type	`OverrideStatus`
	Variation	
	Direction	IN
Parameter	numberOfMessagesToOverride	
	Description	Number of sequential VerifyStatus to override when using a specific counter for authentication verification. This is only considered when OverrideStatus is equal to kSecOcOverrideDropUntilLimit, kSecOcOverrideSkipUntilLimit or kSecOcOverridePassUntilLimit.
	Type	`uint8_t`
	Variation	
	Direction	IN
Enclosing Service Interface	`VerificationStatusConfigurationByFreshnessId`	

⌋

10 Configuration

The configuration model of this functional cluster is defined in [5]. This chapter defines the default values for attributes and semantic constraints for elements specified in [5] that are part of the configuration model of this functional cluster.

10.1 Default Values

This functional cluster does not define any default values for attributes specified in [5].

10.2 Semantic Constraints

This section defines semantic constraints for elements specified in [5] that are part of the configuration model of this functional cluster.

[SWS_CM_CONSTR_00008] Configurable Namespace ⌈Configurable Namespace for CommunicationManagement `ServiceInterface.namespace` shall exist for `ServiceInterface`.⌋

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.

This chapter is generated.

Class	***AbstractIamRemoteSubject*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SCREIAM			
Note	This abstract meta-class defines the proxy information about the remote node. **Tags:** atp.Status=candidate			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Subclasses	IPSecIamRemoteSubject, IpIamRemoteSubject, TlsIamRemoteSubject			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.1: AbstractIamRemoteSubject

Class	***AdaptivePlatformServiceInstance*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a service instance in an abstract way.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Subclasses	*ProvidedApServiceInstance*, *RequiredApServiceInstance*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
e2eEvent ProtectionProps	End2EndEvent ProtectionProps	*	aggr	This aggregation allows to protect an event or a field notifier that is defined inside of the ServiceInterface that is referenced by the ServiceInstance in the role service Interface.
e2eMethod ProtectionProps	End2EndMethod ProtectionProps	*	aggr	This aggregation allows to protect a method or a field getter or a field setter that is defined inside of the Service Interface that is referenced by the ServiceInstance in the role serviceInterface
secureCom Config	ServiceInterface ElementSecureCom Config	*	aggr	Configuration settings to secure the communication of ServiceInterface elements.
serviceInterface Deployment	ServiceInterface Deployment	0..1	ref	Reference to a ServiceInterfaceDeployment that identifies the ServiceInterface that is represented by the Service Instance.

Table A.2: AdaptivePlatformServiceInstance

Class	**Allocator**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the ability to specify an optional custom C++ allocator for a C++ type which may dynamically grow beyond it's initial allocated size during it's lifetime. Any storage principles are defined in the implementation of the allocator itself, which should implement the ISO C++ std::allocator_ traits interface. **Tags:** atp.recommendedPackage=Allocators			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
headerFile	String	0..1	attr	Configuration of the Header File with the custom class declaration
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition of a namespace of an Allocator.

Table A.3: Allocator

Class	**ApApplicationError**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to formally specify the semantics of an application error on the AUTOSAR adaptive platform **Tags:** atp.recommendedPackage=ApplicationErrors			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
errorCode	Integer	0..1	attr	This attribute has the ability to specify the error code value within the enclosing AdaptivePlatformApplication Error.
errorDomain	ApApplicationError Domain	0..1	ref	This reference represents the error domain of the Ap ApplicationError.

Table A.4: ApApplicationError

Class	**ApApplicationErrorDomain**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to define a global error domain for an ApApplicationError. **Tags:** atp.recommendedPackage=ApplicationErrorDomains			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
namespace (ordered)	SymbolProps	*	aggr	This aggregation defines the namespace of the Ap ApplicationErrorDomain
value	PositiveUnlimitedInteger	0..1	attr	This attribute identifies the error category.

Table A.5: ApApplicationErrorDomain

Class	**ApApplicationErrorSet**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class acts as a reference target that represents an entire collection of APApplicationErrors. This takes the burden from ClientServerOperations that reference a larger number of ApApplication Errors. **Tags:** atp.recommendedPackage=ApplicationErrorSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
apApplication Error	ApApplicationError	*	ref	This reference represents the collection of ApApplication Error represented by the enclosing ApApplicationErrorSet

Table A.6: ApApplicationErrorSet

Class	**ApSomeipTransformationProps**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::SerializationProperties			
Note	SOME/IP serialization properties.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *TransformationProps*			
Aggregated by	TransformationPropsSet.transformationProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
alignment	PositiveInteger	0..1	attr	Defines the padding for alignment purposes that will be added by the SOME/IP transformer after the serialized data of the variable data length data element. The alignment shall be specified in Bits.
byteOrder	ByteOrderEnum	0..1	attr	Specifies the byte order of data in the serialized data stream.
implements LegacyString Serialization	Boolean	0..1	attr	This attribute indicates that Strings in the SOME/IP message shall NOT be serialized according to the SOME/IP specification for Strings. If this attribute is set to true, BOM and null-termination shall NOT be added in the serialization for Strings in the payload. If this attribute is set to false (or not set) BOM and null-termination shall be added in the serialization for Strings in the payload according to the SOME/IP specification for Strings. NOTE! This attribute is not future safe, and will be removed in an upcoming AUTOSAR release! **Tags:** atp.Status=obsolete
isDynamic LengthFieldSize	Boolean	0..1	attr	This attribute represents the ability to control the setting of the wire type for TLV encoding. If the attribute is set to true then wire type 5-7 shall be used. If the attribute does not exist or is set to false then wire type 4 shall be used.
sizeOfArray LengthField	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of a variable size Array (Vector), fixed-size Array or an Associative_Map. It describes the size of the length field (in Bytes) that will be put in front of the Array or Associative_Map in the SOME/IP message.

△

Class	ApSomeipTransformationProps			
sizeOfString LengthField	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of a String. It describes the size of the length field (in Bytes) that will be put in front of the String in the SOME/IP message.
sizeOfStruct LengthField	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of an Struct. It describes the size of the length field (in Bytes) that will be put in front of the Struct in the SOME/IP message.
sizeOfUnion LengthField	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of a Union. It describes the size of the length field (in Bytes) that will be put in front of the Union in the SOME/IP message.
sizeOfUnion TypeSelector Field	PositiveInteger	0..1	attr	Configures the SOME/IP serialization for the referenced dataPrototype in case of a Union. It describes the size of the type selector field (in Bytes) that will be put in front of the Union in the SOME/IP message.
stringEncoding	BaseTypeEncoding String	0..1	attr	Configures the encoding for SOME/IP serialization for the referenced dataPrototype in case of an String.

Table A.7: ApSomeipTransformationProps

Class	ApplicationArrayDataType			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which is an array, each element is of the same application data type. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dynamicArray SizeProfile	String	0..1	attr	Specifies the profile which the array will follow if it is a variable size array.
element	ApplicationArray Element	0..1	aggr	This association implements the concept of an array element. That is, in some cases it is necessary to be able to identify single array elements, e.g. as input values for an interpolation routine.

Table A.8: ApplicationArrayDataType

Class	*ApplicationDataType* (abstract)
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes
Note	ApplicationDataType defines a data type from the application point of view. Especially it should be used whenever something "physical" is at stake. An ApplicationDataType represents a set of values as seen in the application model, such as measurement units. It does not consider implementation details such as bit-size, endianess, etc. It should be possible to model the application level aspects of a VFB system by using ApplicationData Types only.
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*
Subclasses	*ApplicationCompositeDataType*, ApplicationPrimitiveDataType
Aggregated by	ARPackage.element

▽

△

Class	**ApplicationDataType** (abstract)			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.9: ApplicationDataType

Class	**ApplicationRecordDataType**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	An application data type which can be decomposed into prototypes of other application data types. **Tags:** atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationCompositeDataType*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
element (ordered)	ApplicationRecordElement	*	aggr	Specifies an element of a record. The aggregation of ApplicationRecordElement is subject to variability with the purpose to support the conditional existence of elements inside a ApplicationrecordDataType. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=element.shortName, element.variationPoint.shortLabel vh.latestBindingTime=preCompileTime

Table A.10: ApplicationRecordDataType

Class	**ApplicationRecordElement**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Describes the properties of one particular element of an application record data type.			
Base	*ARObject*, *ApplicationCompositeElementDataPrototype*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationRecordDataType.element, *AtpClassifier*.atpFeature			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing ApplicationRecordElement as optional. This means the that, at runtime, the ApplicationRecordElement may or may not have a valid value and shall therefore be ignored. The underlying runtime software provides means to set the ApplicationRecordElement as not valid at the sending end of a communication and determine its validity at the receiving end.

Table A.11: ApplicationRecordElement

Class	**ArgumentDataPrototype**
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface
Note	An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.

▽

△

Class	**ArgumentDataPrototype**			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ClientServerOperation.argument			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
direction	ArgumentDirectionEnum	0..1	attr	This attribute specifies the direction of the argument prototype.
serverArgumentImplPolicy	ServerArgumentImplPolicyEnum	0..1	attr	This defines how the argument type of the servers RunnableEntity is implemented. If the attribute is not defined this has the same semantics as if the attribute is set to the value useArgumentType for primitive arguments and structures.

Table A.12: ArgumentDataPrototype

Enumeration	**ArgumentDirectionEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	Use cases: • Arguments in ClientServerOperation can have different directions that need to be formally indicated because they have an impact on how the function signature looks like eventually. • Arguments in BswModuleEntry already determine a function signature, but the direction is used to specify the semantics, especially of pointer arguments.
Aggregated by	ArgumentDataPrototype.direction, SwServiceArg.direction
Literal	***Description***
in	The argument value is passed to the callee. **Tags:** atp.EnumerationLiteralIndex=0
inout	The argument value is passed to the callee but also passed back from the callee to the caller. **Tags:** atp.EnumerationLiteralIndex=1
out	The argument value is passed from the callee to the caller. **Tags:** atp.EnumerationLiteralIndex=2

Table A.13: ArgumentDirectionEnum

Class	***AutosarDataPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of an AutosarDataType.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *DataPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	ArgumentDataPrototype, Field, ParameterDataPrototype, PersistencyDataElement, VariableDataPrototype			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
type	AutosarDataType	0..1	tref	This represents the corresponding data type. **Stereotypes:** isOfType

Table A.14: AutosarDataPrototype

Class	**BaseTypeDirectDefinition**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	*ARObject*, *BaseTypeDefinition*			
Aggregated by	*BaseType*.baseTypeDefinition			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
baseType Encoding	BaseTypeEncoding String	0..1	attr	This specifies, how an object of the current BaseType is encoded, e.g. in an ECU within a message sequence. **Tags:** xml.sequenceOffset=90
baseTypeSize	PositiveInteger	0..1	attr	Describes the length of the data type specified in the container in bits. **Tags:** xml.sequenceOffset=70
byteOrder	ByteOrderEnum	0..1	attr	This attribute specifies the byte order of the base type. **Tags:** xml.sequenceOffset=110
memAlignment	PositiveInteger	0..1	attr	This attribute describes the alignment of the memory object in bits. E.g. "8" specifies, that the object in question is aligned to a byte while "32" specifies that it is aligned four byte. If the value is set to "0" the meaning shall be interpreted as "unspecified". **Tags:** xml.sequenceOffset=100
native Declaration	NativeDeclarationString	0..1	attr	This attribute describes the declaration of such a base type in the native programming language, primarily in the Programming language C. This can then be used by a code generator to include the necessary declarations into a header file. For example BaseType with shortName: "MyUnsignedInt" native Declaration: "unsigned short" Results in typedef unsigned short MyUnsignedInt; If the attribute is not defined the referring Implementation DataTypes will not be generated as a typedef by RTE. If a nativeDeclaration type is given it shall fulfill the characteristic given by basetypeEncoding and baseType Size. This is required to ensure the consistent handling and interpretation by software components, RTE, COM and MCM systems. **Tags:** xml.sequenceOffset=120

Table A.15: BaseTypeDirectDefinition

Enumeration	**ByteOrderEnum**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	When more than one byte is stored in the memory the order of those bytes may differ depending on the architecture of the processing unit. If the least significant byte is stored at the lowest address, this architecture is called little endian and otherwise it is called big endian. ByteOrder is very important in case of communication between different PUs or ECUs.
Aggregated by	ApSomeipTransformationProps.byteOrder, BaseTypeDirectDefinition.byteOrder, DiagnosticCommon Props.defaultEndianness, ISignalToIPduMapping.packingByteOrder, MultiplexedIPdu.selectorField ByteOrder, PduToFrameMapping.packingByteOrder, SegmentPosition.segmentByteOrder, SOMEIP TransformationDescription.byteOrder, System.containerIPduHeaderByteOrder
Literal	***Description***

▽

△

Enumeration	**ByteOrderEnum**
mostSignificantByte First	Most significant byte shall come at the lowest address (also known as BigEndian or as Motorola-Format) **Tags:** atp.EnumerationLiteralIndex=0
mostSignificantByte Last	Most significant byte shall come highest address (also known as LittleEndian or as Intel-Format) **Tags:** atp.EnumerationLiteralIndex=1
opaque	For opaque data endianness conversion has to be configured to Opaque. See AUTOSAR COM Specification for more details. **Tags:** atp.EnumerationLiteralIndex=2

Table A.16: ByteOrderEnum

Class	**ClientServerOperation**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	An operation declared within the scope of a client/server interface.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ApplicationInterface.command, *AtpClassifier*.atpFeature, ClientServerInterface.operation, DiagnosticDataElementInterface.read, DiagnosticDataIdentifierInterface.read, DiagnosticDataIdentifierInterface.write, DiagnosticRoutineInterface.requestResult, DiagnosticRoutineInterface.start, DiagnosticRoutineInterface.stop, PhmRecoveryActionInterface.recovery, ServiceInterface.method			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=argument.shortName, argument.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime
fireAndForget	Boolean	0..1	attr	This attribute defines whether this method is a fire&forget method (true) or not (false).
possibleApError	ApApplicationError	*	ref	This reference identifies AdaptivePlatformApplicationErrors as a possible error raised by the enclosing ClientServerOperation.
possibleApErrorSet	ApApplicationErrorSet	*	ref	This reference represents the ability to refer to an entire group of ApApplicationErrors as one model element instead of having to refer to all the represented ApApplicationErrors separately.

Table A.17: ClientServerOperation

Class	**CmModuleInstantiation**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to define a definition of a Communication Management instantiation. **Tags:** atp.Status=candidate			
Base	*ARObject*, *AdaptiveModuleInstantiation*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *NonOsModuleInstantiation*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, Machine.moduleInstantiation			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	CmModuleInstantiation			
grant	Grant	*	ref	This reference identifies the applicable Grants for this Cm ModuleInstantiation. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=grant atp.Status=candidate
localCom AccessControl Enabled	Boolean	0..1	attr	This switch activates the policy enforcement in Communication Management on local applications. **Tags:** atp.Status=candidate
remoteAccess ControlEnabled	Boolean	0..1	attr	This switch activates the check of the remote subject. **Tags:** atp.Status=candidate

Table A.18: CmModuleInstantiation

Class	ComEventGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to grant access to a ServiceInterface.event. **Tags:** atp.Status=candidate atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *ComGrant*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
design	ComEventGrantDesign	0..1	ref	This reference identifies the ComEventGrantDesign that the enclosing ComEventGrant was created from. **Stereotypes:** atpUriDef **Tags:** atp.Status=candidate
service Deployment	ServiceEvent Deployment	0..1	ref	This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies. **Tags:** atp.Status=candidate

Table A.19: ComEventGrant

Class	ComFieldGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to grant access to a ServiceInterface.field. **Tags:** atp.Status=candidate atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *ComGrant*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
design	ComFieldGrantDesign	0..1	ref	This reference identifies the ComFieldGrantDesign that the enclosing ComFieldGrant was created from. **Stereotypes:** atpUriDef **Tags:** atp.Status=candidate

▽

△

Class	ComFieldGrant			
role	FieldAccessEnum	0..1	attr	This attribute provides the ability to further specify the access to the ServiceInterface.field. **Tags:** atp.Status=candidate
service Deployment	ServiceField Deployment	0..1	ref	This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies. **Tags:** atp.Status=candidate

Table A.20: ComFieldGrant

Class	*ComGrant* (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class serves as the abstract base class for defining specific ComGrants **Tags:** atp.Status=candidate			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Subclasses	ComEventGrant, ComFieldGrant, ComMethodGrant, ComTriggerGrant			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
remoteSubject	AbstractIamRemote Subject	*	ref	This optional reference defines the remoteSubject that is allowed to access the defined Object via the Grant. **Tags:** atp.Status=candidate
serviceInstance	AdaptivePlatform ServiceInstance	0..1	ref	This reference identifies the applicable AdaptivePlatform ServiceInstance for which the grant applies. **Tags:** atp.Status=candidate

Table A.21: ComGrant

Class	ComMethodGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to grant access to a ServiceInterface.method. **Tags:** atp.Status=candidate atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *ComGrant*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
design	ComMethodGrant Design	0..1	ref	This reference identifies the ComMethodGrantDesign that the enclosing ComMethodGrant was created from. **Stereotypes:** atpUriDef **Tags:** atp.Status=candidate
service Deployment	ServiceMethod Deployment	0..1	ref	This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies. **Tags:** atp.Status=candidate

Table A.22: ComMethodGrant

Class	ComOfferServiceGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to grant the offering of a service. **Tags:** atp.Status=candidate atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
design	ComOfferServiceGrant Design	0..1	ref	This reference identifies the ComOfferServiceGrant Design that the enclosing ComOfferServiceGrant was created from. **Stereotypes:** atpUriDef **Tags:** atp.Status=candidate
serviceInstance	AdaptivePlatform ServiceInstance	0..1	ref	This reference identifies the AdaptivePlatformService Instances for which the grant applies. **Tags:** atp.Status=candidate

Table A.23: ComOfferServiceGrant

Class	ComTriggerGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to grant access to a ServiceInterface.trigger **Tags:** atp.Status=candidate atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *ComGrant*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
design	ComTriggerGrant Design	0..1	ref	This reference identifies the ComTriggerGrantDesign that the enclosing ComTriggerGrant was created from **Stereotypes:** atpUriDef **Tags:** atp.Status=candidate
service Deployment	ServiceEvent Deployment	0..1	ref	This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies. **Tags:** atp.Status=candidate

Table A.24: ComTriggerGrant

Class	***CommunicationConnector*** (abstract)
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreTopology
Note	The connection between the referencing ECU and the referenced channel via the referenced controller. Connectors are used to describe the bus interfaces of the ECUs and to specify the sending/receiving behavior. Each CommunicationConnector has a reference to exactly one communicationController. Note: Several CommunicationConnectors can be assigned to one PhysicalChannel in the scope of one ECU Instance.
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*

▽

△

Class	***CommunicationConnector*** (abstract)			
Subclasses	*AbstractCanCommunicationConnector*, EthernetCommunicationConnector, FlexrayCommunication Connector, UserDefinedCommunicationConnector			
Aggregated by	EcuInstance.connector, MachineDesign.communicationConnector			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
commController	Communication Controller	0..1	ref	Reference to the communication controller. The CommunicationConnector and referenced CommunicationController shall be aggregated by the same ECUInstance. The communicationController can be referenced by several CommunicationConnector elements. This is important for the FlexRay Bus. FlexRay communicates via two physical channels. But only one controller in an ECU is responsible for both channels. Thus, two connectors (for channel A and for channel B) shall reference to the same controller.
createEcu WakeupSource	Boolean	0..1	attr	If this parameter is available and set to true then a channel wakeup source shall be created for the Physical Channel referencing this CommunicationConnector.
pncFilterArray Mask (ordered)	PositiveInteger	*	attr	Bit mask for NM-Pdu Payload used to configure the NM filter mask for the Network Management.

Table A.25: CommunicationConnector

Class	**CouplingPort**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	A CouplingPort is used to connect a CouplingElement with an EcuInstance or two CouplingElements with each other via a CouplingPortConnection. Optionally, the CouplingPort may also have a reference to a macMulticastGroup and a defaultVLAN.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	CouplingElement.couplingPort, EthernetCommunicationController.couplingPort			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
connection Negotiation Behavior	EthernetConnection NegotiationEnum	0..1	attr	Specifies the connection negotiation of the CouplingPort.
couplingPort Details	CouplingPortDetails	0..1	aggr	Defines more details of a CouplingPort in case a more specific configuration is required.
couplingPort Role	CouplingPortRoleEnum	0..1	attr	Defines the role this CouplingPort takes in the context of the CouplingElement.
defaultVlan	EthernetPhysical Channel	0..1	ref	The vLanIdentifier of the referenced VLAN is the Default-PVID (port VLAN ID). A Port VLAN ID is a default VLAN ID that is assigned to an access CouplingPort to designate the VLAN segment to which this port is connected. Also, if a CouplingPort has not been configured with any VLAN memberships, the virtual switch's Port VLAN ID (pvid) becomes the default VLAN ID for the ports connection. This identifier/tag is added for incoming untagged messages at the port (ingress tagging). For outgoing messages with this identifier, the tag is removed at the port (egress untagging, depending on the Vlan Membership.sendActivity).

▽

△

Class	**CouplingPort**			
macAddress VlanAssignment	MacAddressVlan Membership	*	aggr	Statically defines the assignment of MAC-Multicast-Addresses, optionally together with VLANs, to this CouplingPort. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=macAddressVlanAssignment.shortName, macAddressVlanAssignment.variationPoint.shortLabel vh.latestBindingTime=postBuild
macLayerType	EthernetMacLayerType Enum	0..1	attr	Specifies the mac layer type of the CouplingPort.
macMulticast Address	MacMulticastGroup	*	ref	Assigns a set of MAC-Multicast-Addresses which are addressable via this CouplingPort. This is a static pre-configuration and further addresses may be learned during runtime. **Tags:** atp.Status=obsolete
macSecProps	MacSecProps	*	aggr	Properties to configure MACsec (Media access control security) and the MKA (MACsec Key Agreement) for the CouplingPort (PHY). **Tags:** atp.Status=candidate
physicalLayer Type	EthernetPhysicalLayer TypeEnum	0..1	attr	Specifies the physical layer type of the CouplingPort.
plcaProps	PlcaProps	0..1	aggr	Optional properties for configuration of PLCA (Physical Layer Collision Avoidance) in case 10-BASE-T1S Ethernet is used and PLCA is enabled on the Coupling Port (PHY).
pncMapping	PncMappingIdent	*	ref	Reference to the partial networks this CouplingPort participates in. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=pncMapping
receiveActivity	EthernetSwitchVlan IngressTagEnum	0..1	attr	Defines the handling of frames at the ingress port.
vlan Membership	VlanMembership	*	aggr	Messages of VLANs that are defined here can be communicated via the CouplingPort.
wakeupSleep OnDataline Config	EthernetWakeupSleep OnDatalineConfig	0..1	ref	Optional reference to EthernetWakeupSleepOnDataline Config.

Table A.26: CouplingPort

Class	***CppImplementationDataType*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a reusable data type definition taken as a the basis for a C++ language binding			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	CustomCppImplementationDataType, StdCppImplementationDataType			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***CppImplementationDataType*** (abstract)			
arraySize	PositiveInteger	0..1	attr	This attribute can be used to specify the array size if the enclosing CppImplementationDataType has array semantics. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
headerFile	String	0..1	attr	Configuration of the Header File with the custom class declaration.
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition an own namespace for the enclosing CppImplementationData Type.
subElement (ordered)	CppImplementation DataTypeElement	*	aggr	This represents the collection of sub-elements of the enclosing CppImplementationDataType
template Argument (ordered)	CppTemplateArgument	*	aggr	This aggregation allows for the specification of properties of template arguments
typeEmitter	NameToken	0..1	attr	This attribute can be taken to control how the respective CppImplementationDataType is contributed to the language binding.
typeReference	CppImplementation DataType	0..1	ref	This reference shall be defined to define a type reference (a.k.a. typedef).

Table A.27: CppImplementationDataType

Class	**CppImplementationDataTypeElement**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	Declares a data object which is locally aggregated. Such an element can only be used within the scope where it is aggregated. A CppImplementationDataTypeElement is used to represent an element of a structure, defining its type.			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, *CppImplementationDataType*.subElement			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
isOptional	Boolean	0..1	attr	This attribute represents the ability to declare the enclosing CppImplementationDataTypeElement as optional. This means the that, at runtime, the Cpp ImplementationDataTypeElement may or may not have a valid value and shall therefore be ignored. The underlying runtime software provides means to set the CppImplementationDataTypeElement as not valid at the sending end of a communication and determine its validity at the receiving end.
swDataDef Props	SwDataDefProps	0..1	aggr	This aggregation allows for the definition of qualifying properties of the enclosing CppImplementationDataType Element. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=swDataDefProps
typeReference	CppImplementation DataTypeElement Qualifier	0..1	aggr	This aggregation defines the type of the Cpp ImplementationDataTypeElement and determines whether in C++ the CppImplementationDataTypeElement is defined inside or outside of the enclosing Cpp ImplementationDataType.

Table A.28: CppImplementationDataTypeElement

Class	**CppTemplateArgument**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class has the ability to define properties for template arguments.			
Base	*ARObject*			
Aggregated by	*CppImplementationDataType*.templateArgument			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
allocator	Allocator	0..1	ref	This reference identifies the applicable allocator.
category	CategoryString	0..1	attr	This attribute shall be used to contribute further clarification regarding the semantics of the enclosing Cpp TemplateArgument.
inplace	Boolean	0..1	attr	This attribute specifies whether the shortName of the referenced templateType is used in the code generation and the type declaration is defined outside of the enclosing CppImplementationDataType (true) or whether the type definition is embedded inside of the enclosing CppImplementationDataType and the shortName is ignored (false).
templateType	CppImplementation DataType	0..1	ref	This reference identifies the data type of the specific template argument required for the language binding.

Table A.29: CppTemplateArgument

Class	***DataPrototype*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	Base class for prototypical roles of any data type.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	*ApplicationCompositeElementDataPrototype*, *AutosarDataPrototype*			
Aggregated by	*AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
swDataDef Props	SwDataDefProps	0..1	aggr	This property allows to specify data definition properties which apply on data prototype level. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=swDataDefProps

Table A.30: DataPrototype

Class	**DataTypeMap**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	This class represents the relationship between ApplicationDataType and its implementing Abstract ImplementationDataType.			
Base	*ARObject*			
Aggregated by	DataTypeMappingSet.dataTypeMap			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
applicationData Type	ApplicationDataType	0..1	ref	This is the corresponding ApplicationDataType
implementation DataType	AbstractImplementation DataType	0..1	ref	This is the corresponding AbstractImplementationData Type.

Table A.31: DataTypeMap

Class	DdsEventDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	DDS configuration settings for an Event.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceEventDeployment*			
Aggregated by	DdsFieldDeployment.notifier, *ServiceInterfaceDeployment*.eventDeployment			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
eventTopic AccessRule	DdsTopicAccessRule	0..1	ref	DDS Security access rule applicable to the DDS Topics used for the service interface event.
topicName	String	0..1	attr	Name of the DDS Topic associated with the Event.
transport Protocol	String	*	attr	This attribute defines over which Transport Layer Protocol(s) this event is intended to be sent.

Table A.32: DdsEventDeployment

Class	DdsEventQosProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	Configuration properties of the Event using DDS as the underlying network binding.			
Base	*ARObject*, *DdsQosProps*			
Aggregated by	DdsProvidedServiceInstance.eventQosProps, DdsRequiredServiceInstance.eventQosProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
event	ServiceEvent Deployment	0..1	ref	Reference to an event that is provided.

Table A.33: DdsEventQosProps

Class	DdsFieldQosProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	Configuration properties of the Field interaction when using DDS as the underlying network binding.			
Base	*ARObject*, *DdsQosProps*			
Aggregated by	DdsProvidedServiceInstance.fieldNotifierQosProps, DdsRequiredServiceInstance.fieldNotifierQosProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
field	ServiceField Deployment	0..1	ref	Reference to the field.

Table A.34: DdsFieldQosProps

Class	DdsProvidedServiceInstance			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a provided service instance in a concrete implementation on top of DDS. **Tags:** atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *DdsQosProps*, *DdsServiceInstanceProps*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *ProvidedApServiceInstance*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
discoveryType	DdsServiceInstance DiscoveryTypeEnum	0..1	attr	Discovery protocol.

△

Class	DdsProvidedServiceInstance			
eventQosProps	DdsEventQosProps	*	aggr	List of configuration properties for the Events that are provided by the Service Instance.
fieldNotifierQosProps	DdsFieldQosProps	*	aggr	List of configuration properties for Field notifiers that are provided by the Service Instance.
resourceIdentifierType	DdsServiceInstanceResourceIdentifierTypeEnum	0..1	attr	Type of resource identification scheme.
serviceInstanceId	PositiveInteger	0..1	attr	Identification number that is used by DDS to identify DomainParticipants associated with an instance of the service.

Table A.35: DdsProvidedServiceInstance

Class	***DdsQosProps*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	QoS configuration properties for the DDS entities associated with an event, method, or field provided by or requested from a Service Instance using DDS as the underlying network binding.			
Base	*ARObject*			
Subclasses	DdsEventQosProps, DdsFieldQosProps, *DdsServiceInstanceProps*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
qosProfile	String	0..1	attr	Identifies a group of QoS Policies that apply to the DDS entities associated with the event, method, field, or the service instance.

Table A.36: DdsQosProps

Class	DdsRequiredServiceInstance			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a required service instance in a concrete implementation on top of DDS. **Tags:** atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *DdsQosProps*, *DdsServiceInstanceProps*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *RequiredApServiceInstance*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blocklistedVersion	DdsServiceVersion	*	aggr	Collection of blocklisted versions.
discoveryType	DdsServiceInstanceDiscoveryTypeEnum	0..1	attr	Discovery protocol.
eventQosProps	DdsEventQosProps	*	aggr	List of configuration properties for the Events that are required by the Service Instance.
fieldNotifierQosProps	DdsFieldQosProps	*	aggr	List of configuration properties for Field notifiers that are required by the Service Instance.
requiredServiceInstanceId	AnyServiceInstanceId	0..1	attr	This attribute represents the ability to describe the required service instance ID.

Table A.37: DdsRequiredServiceInstance

Class	**DdsServiceInstanceProps** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	Common configuration properties for the DDS entities provided by or requested from a Service Instance using DDS as the underlying network binding.			
Base	*ARObject*, *DdsQosProps*			
Subclasses	DdsProvidedServiceInstance, DdsRequiredServiceInstance			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
domainId	Integer	0..1	attr	This attribute identifies the DDS Domain the Service Instance shall join.

Table A.38: DdsServiceInstanceProps

Class	**DdsServiceInterfaceDeployment**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	DDS configuration settings for a ServiceInterface. **Tags:** atp.recommendedPackage=ServiceInterfaceDeployments			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *ServiceInterfaceDeployment*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
fieldReplyTopic Name	String	0..1	attr	Name of the DDS Reply Topic associated with the Field.
fieldRequest TopicName	String	0..1	attr	Name of the DDS Request Topic associated with the Field.
fieldTopics AccessRule	DdsTopicAccessRule	0..1	ref	DDS Security access rule applicable to the DDS Topics used for service interface field access methods (Get, Set).
methodReply TopicName	String	0..1	attr	Name of the DDS Reply Topic associated with the Method.
methodRequest TopicName	String	0..1	attr	Name of the DDS Request Topic associated with the Method.
methodTopics AccessRule	DdsTopicAccessRule	0..1	ref	DDS Security access rule applicable to the DDS Topics used for service interface methods.
serviceInterface Id	String	0..1	attr	Unique Identifier that identifies the ServiceInterface in DDS. This Identifier is encoded in the USER_DATA QoS of the DomainParticipant associated with the Service Instance and its value is propagated by DDS Discovery messages.
transport Protocol	String	*	attr	This attribute defines over which Transport Layer Protocol(s) this Method is intended to be sent.

Table A.39: DdsServiceInterfaceDeployment

Class	**E2EProfileConfiguration**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::E2E			
Note	This element holds E2E profile specific configuration settings.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	E2EProfileConfigurationSet.e2eProfileConfiguration			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
clearFromValid ToInvalid	Boolean	0..1	attr	Clear monitoring window on transition from state Valid to state Invalid.

▽

△

Class	**E2EProfileConfiguration**			
dataIdMode	DataIdModeEnum	0..1	attr	This attribute describes the inclusion mode that is used to include the implicit Data ID in the one-byte CRC.
e2eProfile Compatibility Props	E2EProfileCompatibility Props	0..1	ref	Reference to additional settings for the E2E state machine.
maxDelta Counter	PositiveInteger	0..1	attr	Maximum allowed difference between two counter values of two consecutively received valid messages. For example, if the receiver gets data with counter 1 and Max DeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
maxErrorState Init	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_INIT.
maxErrorState Invalid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_INVALID.
maxErrorState Valid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_VALID.
minOkStateInit	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INIT.
minOkState Invalid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INVALID.
minOkState Valid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_VALID.
profileName	NameToken	0..1	attr	Definition of the E2E profile.
windowSizeInit	PositiveInteger	0..1	attr	Size of the monitoring window of state Init for the E2E state machine.
windowSize Invalid	PositiveInteger	0..1	attr	Size of the monitoring window of state Invalid for the E2E state machine.
windowSize Valid	PositiveInteger	0..1	attr	Size of the monitoring window of state Valid for the E2E state machine.

Table A.40: E2EProfileConfiguration

Class	**End2EndEventProtectionProps**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::E2E			
Note	This element allows to protect an event or a field notifier with an E2E profile.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AdaptivePlatformServiceInstance*.e2eEventProtectionProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataId (ordered)	PositiveInteger	*	attr	This represents a unique numerical identifier for the referenced event or field notifier that is included in the CRC calculation. Note: ID is used for protection against masquerading. The details concerning the maximum number of values (this information is specific for each E2E profile) applicable for this attribute are controlled by a semantic constraint that depends on the category of the EndToEnd Protection.

▽

△

Class	**End2EndEventProtectionProps**			
dataLength	PositiveInteger	0..1	attr	Length of payload including E2E header in bits.
dataUpdate Period	TimeValue	0..1	attr	This attribute describes the period in which the applications are assumed to process E2E-protected messages. The middleware does not use this attribute at all.
e2eProfile Configuration	E2EProfileConfiguration	0..1	ref	Reference to E2E profile configuration settings that are valid to protect the referenced event or field notifier.
event	ServiceEvent Deployment	0..1	ref	Reference to an event that is protected by the E2E profile.
maxDataLength	PositiveInteger	0..1	attr	Maximum length of payload including E2E header in bits.
minDataLength	PositiveInteger	0..1	attr	Minimum length of payload including E2E header in bits.

Table A.41: End2EndEventProtectionProps

Class	**End2EndMethodProtectionProps**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::E2E			
Note	This element allows to protect a method, a field setter or a field getter with an E2E profile.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AdaptivePlatformServiceInstance*.e2eMethodProtectionProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
dataId (ordered)	PositiveInteger	*	attr	This represents a numerical identifier that is included in the CRC calculation. This dataId is used for call and response. Note: ID is used for protection against masquerading. The details concerning the maximum number of values (this information is specific for each E2E profile) applicable for this attribute are controlled by a semantic constraint that depends on the category of the EndToEnd Protection.
dataLength	PositiveInteger	0..1	attr	Length of payload including E2E header in bits.
dataUpdate Period	TimeValue	0..1	attr	This attribute describes the period in which the applications are assumed to process E2E-protected messages. The middleware does not use this attribute at all.
e2eProfile Configuration	E2EProfileConfiguration	0..1	ref	Reference to E2E profile configuration settings that are valid to protect the referenced method, field getter or field setter.
maxDataLength	PositiveInteger	0..1	attr	Maximum length of payload including E2E header in bits.
method	ServiceMethod Deployment	0..1	ref	Reference to a method, a field getter or a field setter that is protected by the E2E profile.
minDataLength	PositiveInteger	0..1	attr	Minimum length of payload including E2E header in bits.
sourceId	PositiveInteger	0..1	attr	This represents a unique numerical identifier identifying the source of a certain transmission. In case of C/S communication, this ID uniquely identifies the client. Note: ID is used for protection against masquerading. The details concerning the maximum number of values (this information is specific for each E2E profile) applicable for this attribute are controlled by a semantic constraint that depends on the category of the EndToEnd Protection.

Table A.42: End2EndMethodProtectionProps

Class	**EndToEndTransformationComSpecProps**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	The class EndToEndTransformationIComSpecProps specifies port specific configuration properties for EndToEnd transformer attributes.			
Base	*ARObject*, *Describable*, *TransformationComSpecProps*			
Aggregated by	ClientComSpec.transformationComSpecProps, *ReceiverComSpec*.transformationComSpecProps, ServerComSpec.transformationComSpecProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clearFromValid ToInvalid	Boolean	0..1	attr	Clear monitoring window on transition from state Valid to state Invalid.
disableEndTo EndCheck	Boolean	0..1	attr	Disables/Enables the E2E check. The E2Eheader is removed from the payload independent from the setting of this attribute.
disableEndTo EndState Machine	Boolean	0..1	attr	Disables the E2EStateMachine (only E2E check functionality is performed)
e2eProfile Compatibility Props	E2EProfileCompatibility Props	0..1	ref	Reference to additional settings for the E2E state machine.
maxDelta Counter	PositiveInteger	0..1	attr	Maximum allowed difference between two counter values of two consecutively received valid messages. For example, if the receiver gets data with counter 1 and Max DeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
maxErrorState Init	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_INIT. The minimum value is 0.
maxErrorState Invalid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_INVALID. The minimum value is 0.
maxErrorState Valid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_VALID. The minimum value is 0.
minOkStateInit	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INIT. The minimum value is 1.
minOkState Invalid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INVALID. The minimum value is 1.
minOkState Valid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_VALID. The minimum value is 1.
windowSizeInit	PositiveInteger	0..1	attr	Size of the monitoring window of state Init for the E2E state machine.
windowSize Invalid	PositiveInteger	0..1	attr	Size of the monitoring window of state Invalid for the E2E state machine.
windowSize Valid	PositiveInteger	0..1	attr	Size of the monitoring window of state Valid for the E2E state machine.

Table A.43: EndToEndTransformationComSpecProps

Class	**EndToEndTransformationDescription**			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	EndToEndTransformationDescription holds these attributes which are profile specific and have the same value for all E2E transformers.			
Base	*ARObject*, *Describable*, *TransformationDescription*			
Aggregated by	TransformationTechnology.transformationDescription			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clearFromValid ToInvalid	Boolean	0..1	attr	Clear monitoring window on transition from state Valid to state Invalid.
counterOffset	PositiveInteger	0..1	attr	Offset of the counter in the Data[] array in bits.
crcOffset	PositiveInteger	0..1	attr	Offset of the CRC in the Data[] array in bits.
dataIdMode	DataIdModeEnum	0..1	attr	This attribute describes the inclusion mode that is used to include the implicit two-byte Data ID in the one-byte CRC.
dataIdNibble Offset	PositiveInteger	0..1	attr	Offset of the Data ID nibble in the Data[] array in bits.
e2eProfile Compatibility Props	E2EProfileCompatibility Props	0..1	ref	Reference to additional settings for the E2E state machine.
maxDelta Counter	PositiveInteger	0..1	attr	Maximum allowed difference between two counter values of two consecutively received valid messages. For example, if the receiver gets data with counter 1 and Max DeltaCounter is 3, then at the next reception the receiver can accept Counters with values 2, 3 or 4.
maxErrorState Init	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_INIT.
maxErrorState Invalid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_INVALID.
maxErrorState Valid	PositiveInteger	0..1	attr	Maximal number of checks in which ProfileStatus equal to E2E_P_ERROR was determined, within the last Window Size checks, for the state E2E_SM_VALID.
maxNoNewOr RepeatedData	PositiveInteger	0..1	attr	The maximum allowed amount of consecutive failed counter checks.
minOkStateInit	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INIT.
minOkState Invalid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_INVALID.
minOkState Valid	PositiveInteger	0..1	attr	Minimal number of checks in which ProfileStatus equal to E2E_P_OK was determined, within the last WindowSize checks, for the state E2E_SM_VALID.
offset	PositiveInteger	0..1	attr	Offset of the E2E header in the Data[] array in bits.
profileBehavior	EndToEndProfile BehaviorEnum	0..1	attr	Behavior of the check functionality
profileName	NameToken	0..1	attr	Definition of the E2E profile.
syncCounterInit	PositiveInteger	0..1	attr	Number of checks required for validating the consistency of the counter that shall be received with a valid counter (i.e. counter within the allowed lock-in range) after the detection of an unexpected behavior of a received counter.

▽

△

Class	EndToEndTransformationDescription			
upperHeader BitsToShift	PositiveInteger	0..1	attr	This attribute describes the number of upper-header bits to be shifted. value = 0 or not present: shift of upper header is NOT performed. value > 0: the E2E Transformer on the protect-side, takes the first upperHeaderBitsToShift bits from the upper buffer (e.g. SOME/IP header part generated by SOME/IP transformer) and shifts them towards the lower bytes and bits within the Data[] for the length of the E2E header (e.g. 12 bytes in case of E2E Profile 4). This means the shift distance is fixed - it depends on the E2E header size - what is configured here is the number of bits that are to be shifted. This option is defined because the Some/IP header generated by SOME/IP transformer shall be, due to compatibility between non-protected and E2E-protected communication, at the same position, which is before E2E header.
windowSizeInit	PositiveInteger	0..1	attr	Size of the monitoring window of state Init for the E2E state machine.
windowSize Invalid	PositiveInteger	0..1	attr	Size of the monitoring window of state Invalid for the E2E state machine.
windowSize Valid	PositiveInteger	0..1	attr	Size of the monitoring window of state Valid for the E2E state machine.

Table A.44: EndToEndTransformationDescription

Class	«atpVariation» **EthernetCluster**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Ethernet-specific cluster attributes. **Tags:** atp.recommendedPackage=CommunicationClusters			
Base	*ARElement*, *ARObject*, *CollectableElement*, *CommunicationCluster*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
couplingPort Connection	CouplingPort Connection	*	aggr	Specification of connections between CouplingElements and EcuInstances. Note: This atpSplitable property has no atp.Splitkey due to atpVariation (PropertySetPattern). **Stereotypes:** atpSplitable; atpVariation **Tags:** vh.latestBindingTime=postBuild
couplingPort StartupActive Time	TimeValue	0..1	attr	The attribute specifies the time in second a coupling port is switched on to enable the host ECU (ECU that maintains an Ethernet switch) to listen to the network for potential network management requests.
couplingPort SwitchoffDelay	TimeValue	0..1	attr	Switch off delay for CouplingPorts in seconds. It denotes the delay of switching off couplingPorts after the request to switch off a couplingPort was issued. (e.g. switch off of Ethernet switch ports).
macMulticast Group	MacMulticastGroup	*	aggr	MacMulticastGroup that is defined for the Subnet (EthernetCluster).

Table A.45: EthernetCluster

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.46: EthernetCommunicationConnector

Class	«atpVariation» **EthernetCommunicationController**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Ethernet specific communication port attributes.			
Base	*ARObject*, *CommunicationController*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	EcuInstance.commController, MachineDesign.communicationController			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
canXlConfig	AbstractCan Communication Controller	0..1	ref	If the Ethernet frames handled by this Ethernet CommunicationController are to be tunneled through CAN XL, then this reference shall refer to the Abstract CanCommunicationController that aggregates the Can ControllerXlConfiguration of the physical CAN XL channel to be used for tunneling.
couplingPort	CouplingPort	*	aggr	Optional CouplingPort that can be used to connect the ECU to a CouplingElement (e.g. a switch).
macLayerType	EthernetMacLayerType Enum	0..1	attr	Specifies the mac layer type of the ethernet controller.
macUnicast Address	MacAddressString	0..1	attr	Media Access Control address (MAC address) that uniquely identifies each EthernetCommunication Controller in the network.
maximum ReceiveBuffer Length	Integer	0..1	attr	Determines the maximum receive buffer length (frame length) in bytes.
maximum TransmitBuffer Length	Integer	0..1	attr	Determines the maximum transmit buffer length (frame length) in bytes.

△

Class	«atpVariation» **EthernetCommunicationController**			
slaveActAs Passive Communication Slave	Boolean	0..1	attr	This attribute specifies if the EcuInstance is acting as a passive communication slave on the connected Physical Channel. This is used for EthernetCommunication Controllers that use Ethernet hardware which supports wake-up and sleep on the network (e.g. Open Alliance TC10 compliant Ethernet hardware).
slaveQualified UnexpectedLink DownTime	TimeValue	0..1	attr	This attribute specifies time when an unexpected link down is evaluated as link down and indicated to the AUTOSAR communication stack.

Table A.47: EthernetCommunicationController

Class	**Field**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the ability to define a piece of data that can be accessed with read and/or write semantics. It is also possible to generate a notification if the value of the data changes.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Aggregated by	ApplicationInterface.attribute, *AtpClassifier*.atpFeature, ServiceInterface.field			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
hasGetter	Boolean	0..1	attr	This attribute controls whether read access is foreseen to this field.
hasNotifier	Boolean	0..1	attr	This attribute controls whether a notification semantics is foreseen to this field.
hasSetter	Boolean	0..1	attr	This attribute controls whether write access is foreseen to this field.

Table A.48: Field

Enumeration	**FieldAccessEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::GrantDesign::ComGrant
Note	This meta-class provides values that qualify access to a field.
Aggregated by	ComFieldGrant.role, ComFieldGrantDesign.role
Literal	***Description***
getter	Access to the getter of the Field. **Tags:** atp.EnumerationLiteralIndex=0
getterSetter	Access to getter and setter of the field **Tags:** atp.EnumerationLiteralIndex=2
setter	Access to the setter of the Field. **Tags:** atp.EnumerationLiteralIndex=1

Table A.49: FieldAccessEnum

Class	**FieldSenderComSpec**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ComSpec
Note	Port specific communication attributes for a Field that is defined in a ServiceInterface.
Base	*ARObject*, *PPortComSpec*, *SenderComSpec*
Aggregated by	*AbstractProvidedPortPrototype*.providedComSpec, PortPrototypeBlueprint.providedComSpec

▽

△

Class	**FieldSenderComSpec**			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Initial value for a Field that is set before the Service Interface is offered.

Table A.50: FieldSenderComSpec

Class	**GlobalTimeDomain**			
Package	M2::AUTOSARTemplates::SystemTemplate::GlobalTime			
Note	This represents the ability to define a global time domain. **Tags:** atp.recommendedPackage=GlobalTimeDomains			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
debounceTime	TimeValue	0..1	attr	Defines the minimum amount of time between two time sync messages are transmitted.
domainId	PositiveInteger	0..1	attr	This represents the ID of the GlobalTimeDomain used in the network messages sent on behalf of global time management.
gateway	GlobalTimeGateway	*	aggr	A GlobalTimeGateway may exist in the context of a GlobalTimeDomain to actively update the global time information as it is routed from one GlobalTimeDomain to another. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=gateway.shortName, gateway.variation Point.shortLabel vh.latestBindingTime=postBuild
globalTime CorrectionProps	GlobalTimeCorrection Props	0..1	aggr	Defintion of attributes for rate and offset correction.
globalTime Domain Property	AbstractGlobalTime DomainProps	0..1	aggr	Additional properties of the GlobalTimeDomain. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=globalTimeDomainProperty, globalTime DomainProperty.variationPoint.shortLabel vh.latestBindingTime=postBuild
globalTime Master	GlobalTimeMaster	0..1	aggr	This represents the single master of a GlobalTime Domain. A GlobalTimeDomain may have no GlobalTime Domain.master, e.g. when it gets its time from a GPS receiver. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=globalTimeMaster.shortName, globalTime Master.variationPoint.shortLabel vh.latestBindingTime=postBuild
globalTimeSub Domain	GlobalTimeDomain	*	ref	By this means it is possible to create a hierarchy of sub Domains where one global time domain can declare one or more other global time domains as its subDomains. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=globalTimeSubDomain.globalTimeDomain, globalTimeSubDomain.variationPoint.shortLabel vh.latestBindingTime=postBuild

▽

△

Class	**GlobalTimeDomain**			
icvFreshness ValueId	PositiveInteger	0..1	attr	This attribute defines the Id of the Freshness Value for the Integrity Check Value (ICV) calculation and verification.
icvSecureCom Props	SecOcSecureCom Props	0..1	ref	Reference to a SecureComProps definition to be used for the Integrity Check Value (ICV) calculation and verification. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=icvSecureComProps.secOcSecureCom Props, icvSecureComProps.variationPoint.shortLabel vh.latestBindingTime=postBuild
maxProgression Mismatch Threshold	TimeValue	0..1	attr	This attribute defines the maximum allowed difference between local time and fallback time of the time base in seconds.
network SegmentId	NetworkSegment Identification	0..1	aggr	Defines the numerical identification of a GlobalTime sub domain.
pduTriggering	PduTriggering	0..1	ref	This PduTriggering will be taken to transmit the global time information from a GlobalTimeMaster to a the associated GlobalTimeSlaves. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=pduTriggering.pduTriggering, pdu Triggering.variationPoint.shortLabel vh.latestBindingTime=postBuild
slave	GlobalTimeSlave	*	aggr	This represents the collections of slaves of the Global TimeDomain. A GlobalTimeDomain may have no Global TimeDomain.slaves, e.g. when it propagates its time directly to sub domains. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=slave.shortName, slave.variationPoint.short Label vh.latestBindingTime=postBuild
syncLoss Timeout	TimeValue	0..1	attr	This attribute describes the timeout for the situation that the time synchronization gets lost in the scope of the time domain.

Table A.51: GlobalTimeDomain

Class	***IEEE1722TpAcfBusPart*** (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::TransportProtocols::IEEE1722Tp::IEEE1722TpAcf			
Note	Definition of one IEEE1722Tp ACF part transported over the IEEE1722Tp channel. **Tags:** atp.Status=candidate			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	IEEE1722TpAcfCanPart, IEEE1722TpAcfLinPart			
Aggregated by	*IEEE1722TpAcfBus*.acfPart			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
collectionTrigger	PduCollectionTrigger Enum	0..1	attr	Defines whether putting this AcfPart to the IEEE1722Tp ACF message triggers immediate sending of the IEEE1722Tp ACF message.

Table A.52: IEEE1722TpAcfBusPart

Class	IEEE1722TpAcfCanPart			
Package	M2::AUTOSARTemplates::SystemTemplate::TransportProtocols::IEEE1722Tp::IEEE1722TpAcf			
Note	Definition of one CAN part (frame or frame range) transported over the IEEE1722Tp channel. **Tags:** atp.Status=candidate			
Base	*ARObject*, *IEEE1722TpAcfBusPart*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*IEEE1722TpAcfBus*.acfPart			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
canAddressing Mode	CanAddressingMode Type	0..1	attr	Defines whether standard or extended address format shall be used.
canBitRate Switch	Boolean	0..1	attr	Defines whether the bit rate switch bit shall be set.
canFrameTx Behavior	CanFrameTxBehavior Enum	0..1	attr	Defines which CAN protocol shall be used for frame transmission.
canIdentifier	PositiveInteger	0..1	attr	Optional Can Id defined in case the Can Id can not be determined during runtime.
canIdentifier Mask	PositiveInteger	0..1	attr	CAN identifier mask which denotes relevant bits in the CAN Identifier. This attribute defines a CAN Identifier range in an alternative way to canIdentifierRange. It identifies the bits of the configured CAN Identifier that must match the received CAN Identifier.
canIdentifier Range	RxIdentifierRange	0..1	aggr	Definition of the identifier range for IEEE1722Tp ACF Can messages. **Tags:** atp.Status=candidate
sdu	PduTriggering	0..1	ref	Reference to the Pdu transported in the IEEE1722Tp channel. **Tags:** atp.Status=candidate

Table A.53: IEEE1722TpAcfCanPart

Class	IEEE1722TpAcfConnection			
Package	M2::AUTOSARTemplates::SystemTemplate::TransportProtocols::IEEE1722Tp			
Note	ACF IEEE1722Tp connection. **Tags:** atp.Status=candidate atp.recommendedPackage=IEEE1722TpConnections			
Base	*ARElement*, *ARObject*, *CollectableElement*, *IEEE1722TpConnection*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
acfMaxTransit Time	TimeValue	0..1	attr	Defines the time offset that is added to the current time at the producer in order to get the "presentation time" (in seconds) when content shall be presented at the consumers.
acfTransported Bus	IEEE1722TpAcfBus	*	aggr	Definition of the transported busses over this ACF connection. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=acfTransportedBus.shortName, acfTransportedBus.variationPoint.shortLabel atp.Status=candidate vh.latestBindingTime=postBuild

▽

△

Class	IEEE1722TpAcfConnection			
collection Threshold	PositiveInteger	0..1	attr	Defines the size threshold in bytes which, when exceeded, triggers the sending of the IEEE1722Tp ACF message, even when the maxium IEEE1722Tp ACF message size has not been reached yet.
collection Timeout	TimeValue	0..1	attr	When this timeout expires the IEEE1722Tp ACF message is triggered for sending. The respective timer is started when the first Pdu is put into the IEEE1722Tp ACF message. Defined in seconds.
mixedBusType Collection	Boolean	0..1	attr	Defines if this ACF-stream is allowed to collect ACF-messages of different bus kinds (i.e. whether it is allowed to collect CAN and LIN ACF-messages in one ACF-stream message).

Table A.54: IEEE1722TpAcfConnection

Class	IEEE1722TpAcfLinPart			
Package	M2::AUTOSARTemplates::SystemTemplate::TransportProtocols::IEEE1722Tp::IEEE1722TpAcf			
Note	Definition of one LIN part transported over the IEEE1722Tp channel. **Tags:** atp.Status=candidate			
Base	*ARObject*, *IEEE1722TpAcfBusPart*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*IEEE1722TpAcfBus*.acfPart			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
linIdentifier	PositiveInteger	0..1	attr	Optional Lin Id defined in case the Lin Id can not be determined during runtime. **Tags:** atp.Status=candidate

Table A.55: IEEE1722TpAcfLinPart

Class	*IEEE1722TpConnection* (abstract)			
Package	M2::AUTOSARTemplates::SystemTemplate::TransportProtocols::IEEE1722Tp			
Note	Definition of the IEEE1722Tp protocol. **Tags:** atp.Status=candidate			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	IEEE1722TpAcfConnection, *IEEE1722TpAvConnection*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
communication Direction	Communication DirectionType	0..1	attr	Communication Direction of the IEEE1722TpConnection. **Tags:** atp.Status=candidate
destinationMac Address	MacAddressString	0..1	attr	Optional definition of the destination MAC address for this stream. If no given then macAddressStreamId is used as destination MAC address. **Tags:** atp.Status=candidate

▽

△

Class	***IEEE1722TpConnection*** (abstract)			
globalTime Domain	GlobalTimeDomain	0..1	ref	Reference to the GlobalTimeDomain this IEEE1722Tp Connection shall be synchronized with. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=globalTimeDomain.globalTimeDomain, globalTimeDomain.variationPoint.shortLabel vh.latestBindingTime=systemDesignTime
macAddress StreamId	MacAddressString	0..1	attr	MAC Address part of the Stream Id. **Tags:** atp.Status=candidate
uniqueStreamId	PositiveInteger	0..1	attr	Unique Id part of the Stream Id. **Tags:** atp.Status=candidate
version	PositiveInteger	0..1	attr	Version of the IEEE1722TP stream. **Tags:** atp.Status=candidate
vlanPriority	PositiveInteger	0..1	attr	Optional definition of the VLAN priority for this stream.

Table A.56: IEEE1722TpConnection

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.57: IPSecConfig

Class	**IPSeclamRemoteSubject**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SCREIAM			
Note	This meta-class defines the proxy information about the remote node in case of IPsec. **Tags:** atp.Status=candidate atp.recommendedPackage=IamRemoteSubjects			
Base	*ARElement*, *ARObject*, *AbstractIamRemoteSubject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
localIpSecRule	IPSecRule	*	ref	This reference is used to describe theRemoteSubjects local IPSecRules. **Tags:** atp.Status=candidate

Table A.58: IPSeclamRemoteSubject

Class	**IPSecRule**			
Package	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
Note	This element defines an IPsec rule that describes communication traffic that is monitored, protected and filtered.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	IPSecConfig.ipSecRule			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
direction	Communication DirectionType	0..1	attr	This attribute defines the direction in which the traffic is monitored. If this attribute is not set a bidirectional traffic monitoring is assumed.
headerType	IPsecHeaderTypeEnum	0..1	attr	Header type specifying the IPsec security mechanism.
ipProtocol	IPsecIpProtocolEnum	0..1	attr	This attribute defines the relevant IP protocol used in the Security Policy Database (SPD) entry.
localCertificate	CryptoService Certificate	*	ref	This reference identifies the applicable certificate used for a local authentication.
localId	String	0..1	attr	This attribute defines how the local participant should be identified for authentication.
localPortRange End	PositiveInteger	0..1	attr	This attribute restricts the traffic monitoring and defines an end value for the local port range. If this attribute is not set then this rule shall be effective for all local ports. Please note that port ranges are currently not supported in the AUTOSAR AP's operating system backend. If AP systems are involved, each IPsec rule may only contain a single port.
localPortRange Start	PositiveInteger	0..1	attr	This attribute restricts the traffic monitoring and defines a start value for the local port range. If this attribute is not set then this rule shall be effective for all local ports. Please note that port ranges are currently not supported in the AUTOSAR AP's operating system backend. If AP systems are involved, each IPsec rule may only contain a single port.
mode	IPsecModeEnum	0..1	attr	This attribute defines the type of the connection.
policy	IPsecPolicyEnum	0..1	attr	An IPsec policy defines the rules that determine which type of IP traffic needs to be secured using IPsec and how that traffic is secured.
preSharedKey	CryptoServiceKey	0..1	ref	This reference identifies the applicable cryptograhic key used for authentication.
priority	PositiveInteger	0..1	attr	This attribute defines the priority of the IPSecRule (SPD entry). The processing of entries is based on priority, starting with the highest priority "0".
remote Certificate	CryptoService Certificate	*	ref	This reference identifies the applicable certificate used for a remote authentication.
remoteId	String	0..1	attr	This attribute defines how the remote participant should be identified for authentication.
remoteIp Address	NetworkEndpoint	*	ref	Definition of the remote NetworkEndpoint. With this reference the connection between the local Network Endpoint and the remote NetworkEndpoint is described on which the traffic is monitored.

△

Class	IPSecRule			
remotePort RangeEnd	PositiveInteger	0..1	attr	This attribute restricts the traffic monitoring and defines an end value for the remote port range. If this attribute is not set then this rule shall be effective for all local ports. Please note that port ranges are currently not supported in the AUTOSAR AP's operating system backend. If AP systems are involved, each IPsec rule may only contain a single port.
remotePort RangeStart	PositiveInteger	0..1	attr	This attribute restricts the traffic monitoring and defines a start value for the remote port range. If this attribute is not set then this rule shall be effective for all local ports. Please note that port ranges are currently not supported in the AUTOSAR AP's operating system backend. If AP systems are involved, each IPsec rule may only contain a single port.

Table A.59: IPSecRule

Class	ISignal			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Signal of the Interaction Layer. The RTE supports a "signal fan-out" where the same System Signal is sent in different SignalIPdus to multiple receivers. To support the RTE "signal fan-out" each SignalIPdu contains ISignals. If the same System Signal is to be mapped into several SignalIPdus there is one ISignal needed for each ISignalToIPduMapping. ISignals describe the Interface between the Precompile configured RTE and the potentially Postbuild configured Com Stack (see ECUC Parameter Mapping). In case of the SystemSignalGroup an ISignal shall be created for each SystemSignal contained in the SystemSignalGroup. **Tags:** atp.recommendedPackage=ISignals			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
data Transformation	DataTransformation	0..1	ref	Optional reference to a DataTransformation which represents the transformer chain that is used to transform the data that shall be placed inside this ISignal. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=dataTransformation.dataTransformation, dataTransformation.variationPoint.shortLabel vh.latestBindingTime=codeGenerationTime

▽

△

Class	**ISignal**			
dataTypePolicy	DataTypePolicyEnum	0..1	attr	With the aggregation of SwDataDefProps an ISignal specifies how it is represented on the network. This representation follows a particular policy. Note that this causes some redundancy which is intended and can be used to support flexible development methodology as well as subsequent integrity checks. If the policy "networkRepresentationFromComSpec" is chosen the network representation from the ComSpec that is aggregated by the PortPrototype shall be used. If the "override" policy is chosen the requirements specified in the PortInterface and in the ComSpec are not fulfilled by the networkRepresentationProps. In case the System Description doesn't use a complete Software Component Description (VFB View) the "legacy" policy can be chosen.
initValue	ValueSpecification	0..1	aggr	Optional definition of a ISignal's initValue in case the System Description doesn't use a complete Software Component Description (VFB View). This supports the inclusion of legacy system signals. This value can be used to configure the Signal's "Init Value". If a full DataMapping exist for the SystemSignal this information may be available from a configured Sender ComSpec and ReceiverComSpec. In this case the initvalues in SenderComSpec and/or ReceiverComSpec override this optional value specification. Further restrictions apply from the RTE specification.
iSignalProps	ISignalProps	0..1	aggr	Additional optional ISignal properties that may be stored in different files. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=iSignalProps
iSignalType	ISignalTypeEnum	0..1	attr	This attribute defines whether this iSignal is an array that results in a UINT8_N / UINT8_DYN ComSignalType in the COM configuration or a primitive type.
length	UnlimitedInteger	0..1	attr	Size of the signal in bits. The size needs to be derived from the mapped VariableDataPrototype according to the mapping of primitive DataTypes to BaseTypes as used in the RTE. Indicates maximum size for dynamic length signals. The ISignal length of zero bits is allowed.
network Representation Props	SwDataDefProps	0..1	aggr	Specification of the actual network representation. The usage of SwDataDefProps for this purpose is restricted to the attributes compuMethod and baseType. The optional baseType attributes "memAllignment" and "byteOrder" shall not be used. The attribute "dataTypePolicy" in the SystemTemplate element defines whether this network representation shall be ignored and the information shall be taken over from the network representation of the ComSpec. If "override" is chosen by the system integrator the network representation can violate against the requirements defined in the PortInterface and in the network representation of the ComSpec. In case that the System Description doesn't use a complete Software Component Description (VFB View) ▽

▽

△

Class	ISignal			
				△ this element is used to configure "ComSignalDataInvalid Value" and the Data Semantics. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=networkRepresentationProps
reception DefaultValue (ordered)	ValueSpecification	*	aggr	Value used to fill data on the receiver side, if less then expected data is received. The value is expected to cover the entire expected ISignal network payload.
systemSignal	SystemSignal	0..1	ref	Reference to the System Signal that is supposed to be transmitted in the ISignal.
timeout Substitution Value	ValueSpecification	0..1	aggr	Defines and enables the ComTimeoutSubstituition for this ISignal.
transformation ISignalProps	TransformationISignal Props	*	aggr	A transformer chain consists of an ordered list of transformers. The ISignal specific configuration properties for each transformer are defined in the TransformationISignalProps class. The transformer configuration properties that are common for all ISignals are described in the TransformationTechnology class. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=transformationISignalProps

Table A.60: ISignal

Class	ISignalIPdu			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Represents the IPdus handled by Com. The ISignalIPdu assembled and disassembled in AUTOSAR COM consists of one or more signals. In case no multiplexing is performed this IPdu is routed to/from the Interface Layer. A maximum of one dynamic length signal per IPdu is allowed. **Tags:** atp.recommendedPackage=Pdus			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *IPdu*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Pdu*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
iPduTiming Specification	IPduTiming	0..1	aggr	Timing specification for Com IPdus (Transmission Modes). This information is mandatory for the sender in a System Extract. This information may be omitted on receivers in a System Extract. atpVariation: The timing of a Pdu can vary. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=iPduTimingSpecification, iPduTiming Specification.variationPoint.shortLabel vh.latestBindingTime=postBuild

▽

△

Class	ISignalIPdu			
iSignalToPdu Mapping	ISignalToIPduMapping	*	aggr	Definition of SignalToIPduMappings included in the Signal IPdu. atpVariation: The content of a PDU can be variable. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=iSignalToPduMapping.shortName, iSignalToPduMapping.variationPoint.shortLabel vh.latestBindingTime=postBuild
unusedBit Pattern	Integer	0..1	attr	AUTOSAR COM and AUTOSAR IPDUM are filling not used areas of an IPDU with this bit-pattern. This attribute is mandatory to avoid undefined behavior. This byte-pattern will be repeated throughout the IPdu.

Table A.61: ISignalIPdu

Class	ISignalToIPduMapping			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	An ISignalToIPduMapping describes the mapping of ISignals to ISignalIPdus and defines the position of the ISignal within an ISignalIPdu.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ISignalIPdu.iSignalToPduMapping, NmPdu.iSignalToIPduMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
iSignal	ISignal	0..1	ref	Reference to a ISignal that is mapped into the ISignal IPdu. Each ISignal contained in the ISignalGroup shall be mapped into an IPdu by an own ISignalToIPduMapping. The references to the ISignal and to the ISignalGroup in an ISignalToIPduMapping are mutually exclusive.
iSignalGroup	ISignalGroup	0..1	ref	Reference to an ISignalGroup that is mapped into the SignalIPdu. If an ISignalToIPduMapping for an ISignal Group is defined, only the UpdateIndicationBitPosition and the transferProperty is relevant. The startPosition and the packingByteOrder shall be ignored. Each ISignal contained in the ISignalGroup shall be mapped into an IPdu by an own ISignalToIPduMapping. The references to the ISignal and to the ISignalGroup in an ISignalToIPduMapping are mutually exclusive.
packingByte Order	ByteOrderEnum	0..1	attr	This parameter defines the order of the bytes of the signal and the packing into the SignalIPdu. The byte ordering "Little Endian" (MostSignificantByteLast), "Big Endian" (MostSignificantByteFirst) and "Opaque" can be selected. For opaque data endianness conversion shall be configured to Opaque. The value of this attribute impacts the absolute position of the signal into the SignalIPdu (see the startPosition attribute description). For an ISignalGroup the packingByteOrder is irrelevant and shall be ignored.

▽

△

Class	**ISignalToIPduMapping**			
startPosition	UnlimitedInteger	0..1	attr	This parameter is necessary to describe the bitposition of a signal within an SignalIPdu. It denotes the least significant bit for "Little Endian" and the most significant bit for "Big Endian" packed signals within the IPdu (see the description of the packingByteOrder attribute). In AUTOSAR the bit counting is always set to "sawtooth" and the bit order is set to "Decreasing". The bit counting in byte 0 starts with bit 0 (least significant bit). The most significant bit in byte 0 is bit 7. Please note that the way the bytes will be actually sent on the bus does not impact this representation: they will always be seen by the software as a byte array. If a mapping for the ISignalGroup is defined, this attribute is irrelevant and shall be ignored.
transferProperty	TransferPropertyEnum	0..1	attr	Defines how the referenced ISignal contributes to the send triggering of the ISignalIPdu.
update IndicationBit Position	UnlimitedInteger	0..1	attr	The UpdateIndicationBit indicates to the receivers that the signal (or the signal group) was updated by the sender. Length is always one bit. The UpdateIndicationBitPosition attribute describes the position of the update bit within the SignalIPdu. For Signals of a ISignalGroup this attribute is irrelevant and shall be ignored. Note that the exact bit position of the updateIndicationBit Position is linked to the value of the attribute packingByte Order because the method of finding the bit position is different for the values mostSignificantByteFirst and most SignificantByteLast. This means that if the value of packingByteOrder is changed while the value of update IndicationBitPosition remains unchanged the exact bit position of updateIndicationBitPosition within the enclosing ISignalIPdu still undergoes a change. This attribute denotes the least significant bit for "Little Endian" and the most significant bit for "Big Endian" packed signals within the IPdu (see the description of the packingByteOrder attribute). In AUTOSAR the bit counting is always set to "sawtooth" and the bit order is set to "Decreasing". The bit counting in byte 0 starts with bit 0 (least significant bit). The most significant bit in byte 0 is bit 7.

Table A.62: ISignalToIPduMapping

Class	**ISignalTriggering**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	A ISignalTriggering allows an assignment of ISignals to physical channels.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*PhysicalChannel*.iSignalTriggering			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
iSignal	ISignal	0..1	ref	This reference shall be used if an ISignal is transported on the PhysicalChannel. This reference forms an XOR relationship with the ISignalTriggering-ISignalGroup reference.
iSignalGroup	ISignalGroup	0..1	ref	This reference shall be used if an ISignalGroup is transported on the PhysicalChannel. This reference forms an XOR relationship with the ISignal Triggering-ISignal reference.

▽

△

Class	**ISignalTriggering**			
iSignalPort	ISignalPort	*	ref	References to the ISignalPort on every ECU of the system which sends and/or receives the ISignal. References for both the sender and the receiver side shall be included when the system is completely defined.

Table A.63: ISignalTriggering

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractFunctionalClusterDesign*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstance Filter*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, AdaptiveSwcInternal Behavior, ApApplicationEndpoint, *ApmcAbstractDefinition*, *ApmcConfigurationElementDef*, *Apmc ContainerElementValue*, ApmcContainerValue, ApmcEnumerationLiteralDef, ApplicationEndpoint, ApplicationError, AppliedStandard, ArtifactChecksum, ArtifactLocator, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BuildAction Entity*, BuildActionEnvironment, Chapter, CheckpointTransition, ClassContentConditional, ClientId Definition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *Comm ConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPortStructuralElement*, CryptoCertificate, CryptoKeySlot, CryptoKeySlotDesign, CryptoKeySlotUsageDesign, CryptoProvider, *CryptoServiceMapping*, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, Data Transformation, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, DdsDomainRange, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, Diagnostic FunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DiagnosticSovd MethodPrimitive, DltApplication, DltArgument, DltMessage, DoIpInterface, DoIpLogicAddress, DoIp LogicalAddress, DoIpNetworkConfigurationDesign, DoIpRoutingActivation, E2EProfileConfiguration, End2EndEventProtectionProps, End2EndMethodProtectionProps, EndToEndProtection, Ethernet WakeupSleepOnDatalineConfig, EventHandler, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMap Element, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForget MethodMapping, FlexrayArTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, Global Supervision, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722TpAcfBus*, *IEEE1722TpAcfBus Part*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacAddressVlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, MemoryUsage, Method Mapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PersistencyDeploymentElement*, *PersistencyInterfaceElement*, *Phm Supervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, ResourceConsumption, ResourceGroup, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSwComposition Prototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecution Context, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInterfaceElementSecureComConfig, *Service MethodDeployment*, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslation Props, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *SpecElementReference*, *StackUsage*, *StateManagementActionItem*, State ManagementActionList, StateManagementStateNotification, *StateManagementStateRequest*, Static SocketConnection, StructuredReq, SupervisionCheckpoint, SupervisionMode, SupervisionMode Condition, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwitchAsynchronous ▽

▽

△

Class	***Identifiable*** (abstract)			
	△ TrafficShaperGroupEntry, SystemMapping, *TimeBaseResource*, *TimingClock*, TimingClockSync Accuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, Timing ModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, TlsJobMapping, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, TransformationTechnology, Trigger, UcmDescription, UcmRetryStrategy, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=adminData xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:** xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:** xml.sequenceOffset=-50
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:** xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:** xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:** xml.attribute=true

Table A.64: Identifiable

Class	InitialSdDelayConfig			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	This element is used to configure the offer behavior of the server and the find behavior on the client.			
Base	*ARObject*			
Aggregated by	SdClientConfig.initialFindBehavior, SdServerConfig.initialOfferBehavior, SomeipSdClientServiceInstanceConfig.initialFindBehavior, SomeipSdServerServiceInstanceConfig.initialOfferBehavior			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
initialDelayMaxValue	TimeValue	0..1	attr	Max Value in seconds to delay randomly the first offer (if aggregated in role initialOfferBehavior by SomeipSdServerServiceInstanceConfig) or the transmission of a find message (if aggregated in role initialFindBehavior by SomeipSdClientServiceInstanceConfig).
initialDelayMinValue	TimeValue	0..1	attr	Min Value in seconds to delay randomly the first offer (if aggregated in role initialOfferBehavior by SomeipSdServerServiceInstanceConfig) or the transmission of a find message (if aggregated in role initialFindBehavior by SomeipSdClientServiceInstanceConfig).
initialRepetitionsBaseDelay	TimeValue	0..1	attr	The base delay for offer repetitions (if aggregated in role initialOfferBehavior by SomeipSdServerServiceInstanceConfig) or find repetitions (if aggregated in role initialFindBehavior by SomeipSdClientServiceInstanceConfig). Successive find messages have an exponential back off delay.
initialRepetitionsMax	PositiveInteger	0..1	attr	Describes the maximum amount of offer repetitions (if aggregated in role initialOfferBehavior by SomeipSdServerServiceInstanceConfig) or the maximum amount of find repetitions (if aggregated in role initialFindBehavior by SomeipSdClientServiceInstanceConfig).

Table A.65: InitialSdDelayConfig

Class	IplamRemoteSubject			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SCREIAM			
Note	This meta-class defines the proxy information about the remote node in case of general IP communication. **Tags:** atp.Status=candidate atp.recommendedPackage=IamRemoteSubjects			
Base	*ARElement*, *ARObject*, *AbstractIamRemoteSubject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
authenticConnectionProps	IplamAuthenticConnectionProps	*	aggr	Definition of IP rules assigned to the IplamRemoteSubject. **Tags:** atp.Status=candidate

Table A.66: IplamRemoteSubject

Class	Ipv4Configuration
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology
Note	Internet Protocol version 4 (IPv4) configuration.
Base	*ARObject*, *NetworkEndpointAddress*

▽

△

Class	**Ipv4Configuration**			
Aggregated by	NetworkEndpoint.networkEndpointAddress			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
assignment Priority	PositiveInteger	0..1	attr	Priority of assignment (1 is highest). If a new address from an assignment method with a higher priority is available, it overwrites the IP address previously assigned by an assignment method with a lower priority.
defaultGateway	Ip4AddressString	0..1	attr	IP address of the default gateway.
dnsServer Address	Ip4AddressString	*	attr	IP addresses of preconfigured DNS servers. **Tags:** xml.namePlural=DNS-SERVER-ADDRESSES
ipAddressKeep Behavior	IpAddressKeepEnum	0..1	attr	Defines the lifetime of a dynamically fetched IP address.
ipv4Address	Ip4AddressString	0..1	attr	IPv4 Address. Notation: 255.255.255.255. The IP Address shall be declared in case the ipv4AddressSource is FIXED and thus no auto-configuration mechanism is used.
ipv4Address Source	Ipv4AddressSource Enum	0..1	attr	Defines how the node obtains its IP address.
networkMask	Ip4AddressString	0..1	attr	Network mask. Notation 255.255.255.255
ttl	PositiveInteger	0..1	attr	Lifespan of data (0..255). The purpose of the TimeToLive field is to avoid a situation in which an undeliverable datagram keeps circulating on a system.

Table A.67: Ipv4Configuration

Class	**Ipv6Configuration**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Internet Protocol version 6 (IPv6) configuration.			
Base	*ARObject*, *NetworkEndpointAddress*			
Aggregated by	NetworkEndpoint.networkEndpointAddress			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
assignment Priority	PositiveInteger	0..1	attr	Priority of assignment (1 is highest). If a new address from an assignment method with a higher priority is available, it overwrites the IP address previously assigned by an assignment method with a lower priority.
defaultRouter	Ip6AddressString	0..1	attr	IP address of the default router.
dnsServer Address	Ip6AddressString	*	attr	IP addresses of pre configured DNS servers. **Tags:** xml.namePlural=DNS-SERVER-ADDRESSES
enableAnycast	Boolean	0..1	attr	This attribute is used to enable anycast addressing (i.e. to one of multiple receivers).
hopCount	PositiveInteger	0..1	attr	The distance between two hosts. The hop count n means that n gateways separate the source host from the destination host (Range 0..255)
ipAddressKeep Behavior	IpAddressKeepEnum	0..1	attr	Defines the lifetime of a dynamically fetched IP address.
ipAddressPrefix Length	PositiveInteger	0..1	attr	IPv6 prefix length defines the part of the IPv6 address that is the network prefix.
ipv6Address	Ip6AddressString	0..1	attr	IPv6 Address. Notation: FFFF:...:FFFF. The IP Address shall be declared in case the ipv6AddressSource is FIXED and thus no auto-configuration mechanism is used.
ipv6Address Source	Ipv6AddressSource Enum	0..1	attr	Defines how the node obtains its IP address.

Table A.68: Ipv6Configuration

Class	MacSecGlobalKayProps			
Package	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
Note	Configuration of the MAC Security Key Agreement Entity properties that are shared by different KaY configurations. **Tags:** atp.Status=candidate atp.recommendedPackage=MacSecGlobalKayProps			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
bypassEther Type	PositiveInteger	0..255	attr	This attribute is used to define EtherTypes that are bypassed by MACsec. The providedEtherType will not be MACsec protected. **Tags:** atp.Status=candidate
bypassVlan	PositiveInteger	0..255	attr	This attribute is used to define VLAN-IDs that are bypassed by MACsec. The provided VLAN-IDs will not be MACsec protected. (VLAN-ID 0 is interpreted as no-VLAN --> Bypass untagged traffic) **Tags:** atp.Status=candidate

Table A.69: MacSecGlobalKayProps

Class	MacSecKayParticipant			
Package	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
Note	This meta-class configures a MKA participant. **Tags:** atp.Status=candidate atp.recommendedPackage=MacSecKayParticipants			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	MacSecParticipantSet.mkaParticipant			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
ckn	CryptoServiceKey	0..1	ref	Reference to the key where the ckn (Connectivity Association key) is stored. **Tags:** atp.Status=candidate
cryptoAlgo Config	MacSecCryptoAlgo Config	0..1	aggr	Cryptography that is used by the MKA Participant. **Tags:** atp.Status=candidate
sak	CryptoServiceKey	0..1	ref	Reference to the key where SAK shall be stored. **Tags:** atp.Status=candidate

Table A.70: MacSecKayParticipant

Class	MacSecLocalKayProps			
Package	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
Note	Configuration of the MAC Security Key Agreement Entity (KaY). **Tags:** atp.Status=candidate			
Base	*ARObject*			
Aggregated by	MacSecProps.macSecKayConfig			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**MacSecLocalKayProps**			
destinationMac Address	MacAddressString	0..1	attr	This attribute defines the destination MAC Address that is used to calculate the ICV (Integrity Check Value). **Tags:** atp.Status=candidate
globalKayProps	MacSecGlobalKay Props	0..1	ref	Reference to properties that are shared between MAC Security Key Agreement Entities. **Tags:** atp.Status=candidate
keyServer Priority	PositiveInteger	0..1	attr	This attribute defines the key-server priority. **Tags:** atp.Status=candidate
mkaParticipant	MacSecKayParticipant	*	ref	Reference to MKA participant settings supported on the CouplingPort. **Tags:** atp.Status=candidate
role	MacSecRoleEnum	0..1	attr	Role of the MAC Security Key Agreement Entity **Tags:** atp.Status=candidate
sourceMac Address	MacAddressString	0..1	attr	This attribute defines the source MAC Address that is used to calculate the ICV (Integrity Check Value). **Tags:** atp.Status=candidate

Table A.71: MacSecLocalKayProps

Class	**MacSecProps**			
Package	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
Note	This meta-class allows to configure MACsec (Media access control security) and the MKA (MACsec Key Agreement) for the CouplingPort (PHY). **Tags:** atp.Status=candidate			
Base	*ARObject*			
Aggregated by	CouplingPort.macSecProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
autoStart	Boolean	0..1	attr	This attribute defines how the Port Access Entity (PAE) is started: • true := Autostart • false := Manual Start **Tags:** atp.Status=candidate
macSecKay Config	MacSecLocalKayProps	0..1	aggr	Properties to configure the MKA instance (KaY) for a controlled CouplingPort (PaE). **Tags:** atp.Status=candidate
onFail Permissive Mode	MacSecFailPermissive ModeEnum	0..1	attr	This attribute sets the behavior of the Port Access Entity in case MACsec does not succeed. **Tags:** atp.Status=candidate
onFail Permissive ModeTimeout	TimeValue	0..1	attr	Timeout in seconds to enable the controlled port in case onFailPermissiveMode is set to Timeout. **Tags:** atp.Status=candidate
sakRekeyTime Span	TimeValue	0..1	attr	Time in seconds to trigger the rekey of an in use SAK (Static Secure Association key). If set to 0, the rekey will not be triggered after a time span. **Tags:** atp.Status=candidate

Table A.72: MacSecProps

Class	**Machine**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SubSystemDesign::MachineManifest			
Note	Machine that represents an Adaptive Autosar Software Stack. **Tags:** atp.recommendedPackage=Machines			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeployment Element*, *UploadablePackageElement*			
Aggregated by	ARPackage.element, *AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
default Application Timeout	EnterExitTimeout	0..1	aggr	This aggregation defines a default timeout in the context of a given Machine with respect to the launching and termination of applications.
environment Variable	TagWithOptionalValue	*	aggr	This aggregation represents the collection of environment variables that shall be added to the environment defined on the level of the enclosing Machine. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=environmentVariable
machineDesign	MachineDesign	0..1	ref	Reference to the MachineDesign this Machine is implementing.
module Instantiation	AdaptiveModule Instantiation	*	aggr	Configuration of Adaptive Autosar module instances that are running on the machine. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=moduleInstantiation.shortName
processor	Processor	*	aggr	This represents the collection of processors owned by the enclosing machine.
secure Communication Deployment	SecureCommunication Deployment	*	aggr	Target-configuration of secure communication protocol configuration settings to crypto module entities. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=secureCommunication Deployment.shortName
trustedPlatform Executable LaunchBehavior	TrustedPlatform ExecutableLaunch BehaviorEnum	0..1	attr	This attribute controls the behavior of how authentication affects the ability to launch for each Executable.

Table A.73: Machine

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.74: NetworkEndpoint

Enumeration	**PduCollectionTriggerEnum**
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances
Note	Defines whether a Pdu contributes to the triggering of the data transmission if Pdu collection is enabled.
Aggregated by	ContainedIPduProps.trigger, *IEEE1722TpAcfBusPart*.collectionTrigger, SocketConnectionIpdu Identifier.pduCollectionTrigger, SoConIPduIdentifier.pduCollectionTrigger
Literal	***Description***
always	Pdu will trigger the transmission of the data. **Tags:** atp.EnumerationLiteralIndex=0
never	Pdu will be buffered and will not trigger the transmission of the data. **Tags:** atp.EnumerationLiteralIndex=1

Table A.75: PduCollectionTriggerEnum

Class	«atpPrototype» **PduToFrameMapping**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	A PduToFrameMapping defines the composition of Pdus in each frame.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*Frame*.pduToFrameMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
packingByte Order	ByteOrderEnum	0..1	attr	This attribute defines the order of the bytes of the Pdu and the packing into the Frame. Please consider that [constr_3246] and [constr_3222] are restricting the usage of this attribute.
pdu	Pdu	0..1	ref	Reference to a I-Pdu, N-Pdu or NmPdu that is transmitted in the Frame.
startPosition	Integer	0..1	attr	This attribute describes the bitposition of a Pdu within a Frame. Please note that the absolute position of the Pdu in the Frame is determined by the definition of the packingByte Order attribute. If Big Endian is specified, the start position indicates the bit position of the most significant bit in the Frame. If Little Endian is specified, the start position indicates the bit position of the least significant bit in the Frame. The bit counting in byte 0 starts with bit 0 (least significant bit). The most significant bit in byte 0 is bit 7. The Pdus are byte aligned in a Frame and only the values 0, 8, 16, 24,... (for little endian) and 7, 15, 23, ... (for big endian) are allowed.
update IndicationBit Position	Integer	0..1	attr	Indication to the receivers that the corresponding Pdu was updated by the sender. This attribute describes the position of the update bit in the frame that aggregates this PDUToFrameMapping. Length is always one bit. Note that the exact bit position of the updateIndicationBit Position is linked to the value of the attribute packingByte Order because the method of finding the bit position is different for the values mostSignificantByteFirst and most SignificantByteLast. This means that if the value of packingByteOrder is changed while the value of update IndicationBitPosition remains unchanged the exact bit position of updateIndicationBitPosition within the enclosing Frame still undergoes a change. This attribute denotes the least significant bit for "Little Endian" and the most significant bit for "Big Endian" packed signals within the IPdu (see the description of the ▽

▽

△

Class	«atpPrototype» **PduToFrameMapping**			
				△ packingByteOrder attribute). In AUTOSAR the bit counting is always set to "sawtooth" and the bit order is set to "Decreasing". The bit counting in byte 0 starts with bit 0 (least significant bit). The most significant bit in byte 0 is bit 7.

Table A.76: PduToFrameMapping

Class	**PduTriggering**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	The PduTriggering describes on which channel the IPdu is transmitted. The Pdu routing by the PduR is only allowed for subclasses of IPdu. Depending on its relation to entities such channels and clusters it can be unambiguously deduced whether a fan-out is handled by the Pdu router or the Bus Interface. If the fan-out is specified between different clusters it shall be handled by the Pdu Router. If the fan-out is specified between different channels of the same cluster it shall be handled by the Bus Interface.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*PhysicalChannel*.pduTriggering			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
iPdu	Pdu	0..1	ref	Reference to the Pdu for which the PduTriggering is defined. One I-Pdu can be triggered on different channels (PduR fan-out). The Pdu routing by the PduR is only allowed for subclasses of IPdu. Nevertheless is the reference to the Pdu element necessary since the PduTriggering element is also used to specify the sending and receiving connections to Ecu Ports.
iPduPort	IPduPort	*	ref	References to the IPduPort on every ECU of the system which sends and/or receives the I-PDU. References for both the sender and the receiver side shall be included when the system is completely defined.
iSignal Triggering	ISignalTriggering	*	ref	This reference provides the relationship to the ISignal Triggerings that are implemented by the PduTriggering. The reference is optional since no ISignalTriggering can be defined for DCM and Multiplexed Pdus. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=iSignalTriggering.iSignalTriggering, iSignal Triggering.variationPoint.shortLabel vh.latestBindingTime=postBuild
secOcCrypto Mapping	SecOcCryptoService Mapping	0..1	ref	This reference identifies the crypto profile applicable to the usage (send, receive) of the also referenced Secured IPdu. Obviously, this reference is only applicable if the Pdutriggering also references a SecuredIPdu in the role i Pdu.
triggerIPduSend Condition	TriggerIPduSend Condition	*	aggr	Defines the trigger for the Com_TriggerIPDUSend API call. Only if all defined TriggerIPduSendConditions evaluate to true (AND associated) the Com_Trigger IPDUSend API shall be called.

Table A.77: PduTriggering

Class	***PortInterface*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractRawDataStreamInterface*, *AbstractSynchronizedTimeBaseInterface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, FirewallStateSwitchInterface, *IdsmAbstractPort Interface*, LogAndTraceInterface, ModeSwitchInterface, NetworkManagementPortInterface, *Persistency Interface*, *PlatformHealthManagementInterface*, ServiceInterface, *StateManagementPortInterface*, TriggerInterface			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
namespace (ordered)	SymbolProps	*	aggr	This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=namespace.shortName

Table A.78: PortInterface

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. This attribute is used to support the modeling of execution dependencies that utilize the condition of process state. Please note that the process states may not be modeled arbitrarily at any stage of the AUTOSAR workflow because the supported states are standardized in the context of the SWS Execution Management [32].
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations.

Table A.79: Process

Class	***ProvidedApServiceInstance*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a provided service instance in an abstract way.			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *Uploadable PackageElement*			
Subclasses	DdsProvidedServiceInstance, ProvidedSomeipServiceInstance, ProvidedUserDefinedServiceInstance			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.80: ProvidedApServiceInstance

Class	**ProvidedSomeipServiceInstance**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a provided service instance in a concrete implementation on top of SOME/IP. **Tags:** atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *ProvidedApServiceInstance*, *Referrable*, *Uploadable DesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
capability Record (ordered)	TagWithOptionalValue	*	aggr	A sequence of records to store arbitrary name/value pairs conveying additional information about the named service.
eventProps	SomeipEventProps	*	aggr	Configuration settings for individual events that are provided by the ServiceInstance.
loadBalancing Priority	PositiveInteger	0..1	attr	This attribute is used to specify the priority in the load balancing option of SOME/IP that is added to the Offer Service. When a client searches for all service instances of a service, the client shall choose the service instance with highest priority if one is defined.
loadBalancing Weight	PositiveInteger	0..1	attr	This attribute is used to specify the weight in the load balancing option of SOME/IP that is added to the Offer Service. When a client searches for all service instances of a service, the client shall choose the service instance with highest priority if one is defined. If several service instances exist with the highest priority the service instance shall be chosen based on the weights of the service instances.
method ResponseProps	SomeipMethodProps	*	aggr	Configuration settings for individual methods that are provided by the ServiceInstance.
priority	PositiveInteger	0..1	attr	This attribute defines the VLAN frame priority for SOME/IP messages that are resulting from this ProvidedSomeip ServiceInstance (Method and Event communication). Values from 0 (best effort) to 7 (highest) are allowed.
providedEvent Group	SomeipProvidedEvent Group	*	aggr	List of EventGroups that are provided by the Service Instance.
sdServerConfig	SomeipSdServer ServiceInstanceConfig	0..1	ref	Server specific configuration settings relevant for the SOME/IP service discovery.

▽

△

Class	ProvidedSomeipServiceInstance			
serviceInstanceId	PositiveInteger	0..1	attr	Identification number that is used by SOME/IP service discovery to identify the instance of the service. The value 65535 for service instance id is reserved and should not be used.

Table A.81: ProvidedSomeipServiceInstance

Class	ProvidedUserDefinedServiceInstance			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a provided service instance in a concrete implementation that is not standardized by AUTOSAR. **Tags:** atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *ProvidedApServiceInstance*, *Referrable*, *Uploadable DesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table A.82: ProvidedUserDefinedServiceInstance

Class	*Referrable* (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, BswVariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescriptionEntity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, PncMappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, SomeipRequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	Type	Mult.	Kind	Note
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100
shortNameFragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table A.83: Referrable

Class	RequestResponseDelay
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances
Note	Time to wait before answering the query.
Base	*ARObject*

▽

△

Class	RequestResponseDelay			
Aggregated by	SdClientConfig.requestResponseDelay, SdServerConfig.requestResponseDelay, SomeipSdClientEventGroupTimingConfig.requestResponseDelay, SomeipSdServerEventGroupTimingConfig.requestResponseDelay, SomeipSdServerServiceInstanceConfig.requestResponseDelay			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
maxValue	TimeValue	0..1	attr	Maximum allowable response delay to entries received by multicast in seconds.
minValue	TimeValue	0..1	attr	Minimum allowable response delay to entries received by multicast in seconds.

Table A.84: RequestResponseDelay

Class	***RequiredApServiceInstance*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a required service instance in an abstract way.			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Subclasses	DdsRequiredServiceInstance, RequiredSomeipServiceInstance, RequiredUserDefinedServiceInstance			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.85: RequiredApServiceInstance

Class	RequiredSomeipServiceInstance			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a required service instance in a concrete implementation on top of SOME/IP. **Tags:** atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *RequiredApServiceInstance*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
blocklistedVersion	SomeipServiceVersion	*	aggr	Collection of blocklisted versions.
capabilityRecord (ordered)	TagWithOptionalValue	*	aggr	A sequence of records to store arbitrary name/value pairs conveying additional information about the named service.
methodRequestProps	SomeipMethodProps	*	aggr	Configuration settings for individual methods that are requested by the ServiceInstance.
requiredEventGroup	SomeipRequiredEventGroup	*	aggr	List of EventGroups that are used by the RequiredServiceInstance.
requiredMinorVersion	AnyVersionString	0..1	attr	This attribute is used to configure for which minor version of the SomeIp ServiceInterface the Service Discovery will search. Value can be set to a number that represents the Minor Version of the searched service or to ANY.
requiredServiceInstanceId	AnyServiceInstanceId	0..1	attr	This attribute represents the ability to describe the required service instance ID.

▽

△

Class	RequiredSomeipServiceInstance			
sdClientConfig	SomeipSdClientService InstanceConfig	0..1	ref	Client specific configuration settings relevant for the SOME/IP service discovery.
versionDriven FindBehavior	ServiceVersion AcceptanceKindEnum	0..1	attr	Defines the service discovery find behavior.

Table A.86: RequiredSomeipServiceInstance

Class	RequiredUserDefinedServiceInstance			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class represents the ability to describe the existence and configuration of a required service instance in a concrete implementation that is not standardized by AUTOSAR. **Tags:** atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *RequiredApServiceInstance*, *Uploadable DesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table A.87: RequiredUserDefinedServiceInstance

Class	SecOcSecureComProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	Configuration of AUTOSAR SecOC. **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
authentication	CryptoServicePrimitive	0..1	ref	This reference defines the authentication algorithm used for MAC generation and verification.
authentication BuildAttempts	PositiveInteger	0..1	attr	This attribute defines the additional number of authentication build attempts that are to be carried out when the generation of the authentication information failed for a given message. If zero is set then only one authentication attempt is done.
authentication VerifyAttempts	PositiveInteger	0..1	attr	This attribute defines the additional number of authentication attempts that are to be carried out when the generation of the authentication information failed for a given message. If zero is set than only one authentication attempt is done.
authInfoTx Length	PositiveInteger	0..1	attr	This attribute defines the length in bits of the authentication code to be included in the payload of the authenticated Message.
freshnessValue Length	PositiveInteger	0..1	attr	This attribute defines the complete length in bits of the Freshness Value.
freshnessValue TxLength	PositiveInteger	0..1	attr	This attribute defines the length in bits of the Freshness Value to be included in the payload of the secured message.
jobRequirement	SecOcJobRequirement	*	aggr	Collection of cryptographic job requirements.

Table A.88: SecOcSecureComProps

Class	**SecureComProps** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	This meta-class defines a communication security protocol and its configuration settings.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Subclasses	DdsSecureComProps, SecOcSecureComProps, TlsSecureComProps			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.89: SecureComProps

Class	**SecureCommunicationAuthenticationProps**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Authentication properties used to configure SecuredIPdus.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	SecureCommunicationPropsSet.authenticationProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
authInfoTx Length	PositiveInteger	0..1	attr	This attribute defines the length in bits of the authentication code to be included in the payload of the authenticated Pdu.

Table A.90: SecureCommunicationAuthenticationProps

Class	**SecureCommunicationFreshnessProps**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Freshness properties used to configure SecuredIPdus.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	SecureCommunicationPropsSet.freshnessProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
freshness CounterSync Attempts	PositiveInteger	0..1	attr	This attribute defines the number of Freshness Counter re-synchronization attempts when a verification failed for a Secured I-PDU. If the value is zero, there will be no additional verification attempt to synchronize with a potentially better fitting Freshness Counter value. This attribute is only applicable if useFreshnessTimestamp is FALSE.
freshness TimestampTime PeriodFactor	PositiveInteger	0..1	attr	This attribute defines a factor that specifies the time period for the Freshness Timestamp. It holds a multiplication factor that specifies the concrete meaning of a Freshness Timestamp increment by one on basis of microseconds.
freshnessValue Length	PositiveInteger	0..1	attr	This attribute defines the complete length in bits of the Freshness Value. As long as the key doesn't change the counter shall not overflow. The length of the counter shall be determined based on the expected life time of the corresponding key and frequency of usage of the counter.
freshnessValue TxLength	PositiveInteger	0..1	attr	This attribute defines the length in bits of the Freshness Value to be included in the payload of the Secured I-PDU. This length is specific to the least significant bits of the complete Freshness Counter. If the attribute is 0 no Freshness Value is included in the Secured I-PDU.

△

Class	SecureCommunicationFreshnessProps			
useFreshness Timestamp	Boolean	0..1	attr	This attribute specifies whether the Freshness Value is generated through individual Freshness Counters or by a Timestamps. The value is set to TRUE when Timestamps are used.

Table A.91: SecureCommunicationFreshnessProps

Class	SecureCommunicationProps			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	This meta-class contains configuration settings that are specific for an individual SecuredIPdu.			
Base	*ARObject*			
Aggregated by	SecuredIPdu.secureCommunicationProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
authData Freshness Length	PositiveInteger	0..1	attr	This attribute defines the length in bits of the authentic PDU data that is passed to the SWC that verifies and generates the Freshness.
authData FreshnessStart Position	PositiveInteger	0..1	attr	This value determines the start position in bits of the Authentic PDU that shall be passed on to the SWC that verifies and generates the Freshness. The bit counting is done according to TPS_SYST_01068.
authentication BuildAttempts	PositiveInteger	0..1	attr	This attribute specifies the number of authentication build attempts.
authentication Retries	PositiveInteger	0..1	attr	This attribute defines the additional number of authentication attempts that are to be carried out when the generation of the authentication information failed for a given SecuredIPdu. If zero is set than only one authentication attempt is done.
dataId	PositiveInteger	0..1	attr	This attribute defines a numerical identifier for the Secured I-PDU.
freshnessValue Id	PositiveInteger	0..1	attr	This attribute defines the Id of the Freshness Value. The Freshness Value might be a normal counter or a time value.
messageLink Length	PositiveInteger	0..1	attr	SecOC links an AuthenticIPdu and CryptographicIPdu together by repeating a specific part (Message Linker) of the AuthenticIPdu in the CryptographicIPdu. This attribute defines the length in bits of the messageLinker.
messageLink Position	PositiveInteger	0..1	attr	SecOC links an AuthenticIPdu and CryptographicIPdu together by repeating a specific part (Message Linker) of the AuthenticIPdu in the CryptographicIPdu. This attribute defines the startPosition in bits of the messageLinker.
secondary FreshnessValue Id	PositiveInteger	0..1	attr	This attribute defines the Id of the Secondary Freshness Value. The Secondary Freshness Value might be a normal counter or a time value. Please note that this attribute is for documentation only to allow the configuration of required freshness value manager and no upstream mapping is defined for it.
securedArea Length	PositiveInteger	0..1	attr	This attribute defines the length in bytes of the area within the payload Pdu which will be secured.
securedArea Offset	PositiveInteger	0..1	attr	This attribute defines the start position (offset in byte) of the area within the payload Pdu which will be secured.

Table A.92: SecureCommunicationProps

Class	**SecuredIPdu**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	If useAsCryptographicPdu is not set or set to false this IPdu contains the payload of an Authentic IPdu supplemented by additional Authentication Information (Freshness Counter and an Authenticator). If useAsCryptographicPdu is set to true this IPdu contains the Authenticator for a payload that is transported in a separate message. The separate Authentic IPdu is described by the Pdu that is referenced with the payload reference from this SecuredIPdu. **Tags:** atp.recommendedPackage=Pdus			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FibexElement*, *IPdu*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Pdu*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
authentication Props	SecureCommunication AuthenticationProps	0..1	ref	Reference to authentication properties that are valid for this SecuredIPdu.
dynamic RuntimeLength Handling	Boolean	0..1	attr	Defines whether the length information for handling this SecuredIPdu with SecuredIPdu.useSecuredPdu Header=noHeader is taken from the configuration or from the actually provided length information during runtime. true: SecuredIPdu length information is taken from the actually provided length information during runtime. false: SecuredIPdu length information is taken from the configuration.
freshnessProps	SecureCommunication FreshnessProps	0..1	ref	Reference to freshness properties that are valid for this SecuredIPdu.
payload	PduTriggering	0..1	ref	Reference to a Pdu that will be protected against unauthorized manipulation and replay attacks.
secure Communication Props	SecureCommunication Props	0..1	aggr	Specific configuration properties for this SecuredIPdu.
useAs Cryptographic IPdu	Boolean	0..1	attr	If this attribute is set to true the SecuredIPdu contains the Authentication Information for an AuthenticIPdu that is transmitted in a separate message. The AuthenticIPdu contains the original payload, i.e. the secured data. If this attribute is set to false this SecuredIPdu contains the payload of an Authentic IPdu supplemented by additional Authentication Information.
useSecuredPdu Header	SecuredPduHeader Enum	0..1	attr	This attribute defines the size of the header which is inserted into the SecuredIPdu. If this attribute is set to anything but noHeader, the SecuredIPdu contains the Secured I-PDU Header to indicate the length of the AuthenticIPdu. The AuthenticIPdu contains the original payload, i.e. the secured data.

Table A.93: SecuredIPdu

Enumeration	**SerializationTechnologyEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment
Note	This enumeration allows to choose a Serialization Technology.
Aggregated by	SomeipEventDeployment.serializer
Literal	***Description***
signalBased	Signal-Based serializer. **Tags:** atp.EnumerationLiteralIndex=1
someip	SOME/IP Serializer **Tags:** atp.EnumerationLiteralIndex=0

Table A.94: SerializationTechnologyEnum

Class	***ServiceEventDeployment*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	This abstract meta-class represents the ability to specify a deployment of an Event to a middleware transport layer.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DdsEventDeployment, SomeipEventDeployment, UserDefinedEventDeployment			
Aggregated by	*ServiceInterfaceDeployment*.eventDeployment			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
event	VariableDataPrototype	0..1	ref	Reference to an Event that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef
trigger	Trigger	0..1	ref	Reference to a Trigger that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef

Table A.95: ServiceEventDeployment

Class	***ServiceFieldDeployment*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	This abstract meta-class represents the ability to specify a deployment of a Field to a middleware transport layer.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DdsFieldDeployment, SomeipFieldDeployment, UserDefinedFieldDeployment			
Aggregated by	*ServiceInterfaceDeployment*.fieldDeployment			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
field	Field	0..1	ref	Reference to a Field that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef

Table A.96: ServiceFieldDeployment

Class	***ServiceInstanceToMachineMapping*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceMapping			
Note	This meta-class represents the ability to map one or several AdaptivePlatformServiceInstances to a CommunicationConnector of a Machine.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Subclasses	DdsServiceInstanceToMachineMapping, SomeipServiceInstanceToMachineMapping, UserDefined ServiceInstanceToMachineMapping			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
communication Connector	Communication Connector	0..1	ref	Reference to the Machine to which the ServiceInstance is mapped.
secOcCom PropsFor Multicast	SecOcSecureCom Props	*	ref	Reference to communication security configuration settings that are valid for the udp multicast endpoint (Port + Multicast IP Address) defined by the ServiceInstanceTo MachineMapping.

▽

△

Class	***ServiceInstanceToMachineMapping*** (abstract)			
secureCom PropsForTcp	SecureComProps	0..1	ref	Reference to communication security configuration settings that are valid for the tcp unicast endpoint (Tcp Port + Unicast IP Address) defined by the Service InstanceToMachineMapping.
secureCom PropsForUdp	SecureComProps	0..1	ref	Reference to communication security configuration settings that are valid for the udp unicast endpoint (Udp Port + Unicast IP Address) defined by the Service InstanceToMachineMapping.
serviceInstance	AdaptivePlatform ServiceInstance	*	ref	Reference to a ServiceInstance that is mapped to the Machine.

Table A.97: ServiceInstanceToMachineMapping

Class	**ServiceInstanceToPortPrototypeMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceMapping			
Note	This meta-class represents the ability to assign a transport layer dependent ServiceInstance to a Port Prototype. With this mapping it is possible to define how specific PortPrototypes are represented in the middleware in terms of service configuration. **Tags:** atp.recommendedPackage=ServiceInstanceToPortPrototypeMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
portPrototype	PortPrototype	0..1	iref	Reference to a specific PortPrototype that represents the ServiceInstance. **Stereotypes:** atpUriDef **InstanceRef implemented by:** PortPrototypeIn ExecutableInstanceRef
process	Process	0..1	ref	Reference to the Process in which the enclosing Service InstanceToPortPrototypeMapping is executed. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process
processDesign	ProcessDesign	0..1	ref	Reference to the ProcessDesign in which the Executable that contains the SoftwareComponent and the referenced PortPrototype is executed. **Stereotypes:** atpUriDef
serviceInstance	AdaptivePlatform ServiceInstance	0..1	ref	Reference to a ServiceInstance that is represented in the Software Component by the mapped group of Port Prototypes.

Table A.98: ServiceInstanceToPortPrototypeMapping

Class	**ServiceInstanceToSignalMapping**
Package	M2::AUTOSARTemplates::AdaptivePlatform::SignalBasedCommunication
Note	This meta-class is defined for a specific ServiceInstance and contains the mappings of elements of a ServiceInterface for which the ServiceInstance is defined to individual ISignalTriggerings. **Tags:** atp.Status=candidate atp.recommendedPackage=ServiceInstanceToSignalMapping

▽

△

Class	**ServiceInstanceToSignalMapping**			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
eventElement Mapping	SignalBasedEvent ElementToISignal TriggeringMapping	*	aggr	Mapping of an event or an element inside of the event to an ISignalTriggering. **Tags:** atp.Status=candidate
fieldMapping	SignalBasedFieldToI SignalTriggering Mapping	*	aggr	Mapping of a field to ISignalTriggerings. **Tags:** atp.Status=candidate
fireAndForget MethodMapping	SignalBasedFireAnd ForgetMethodToISignal TriggeringMapping	*	aggr	Mapping of an ISignalTriggering being part of a fire and forget message to a ClientServerOperation. **Tags:** atp.Status=candidate
serviceInstance	AdaptivePlatform ServiceInstance	0..1	ref	Reference to a ServiceInstance from which the corresponding ServiceInterface elements will be transported in the signal-based way over a communication medium. **Tags:** atp.Status=candidate
triggerMapping	SignalBasedTriggerToI SignalTriggering Mapping	*	aggr	Mapping of a trigger to an ISignalTriggering. **Tags:** atp.Status=candidate

Table A.99: ServiceInstanceToSignalMapping

Class	**ServiceInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. **Tags:** atp.recommendedPackage=ServiceInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
event	VariableDataPrototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=event.shortName, event.variationPoint.short Label vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
field	Field	*	aggr	This represents the collection of fields defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=field.shortName, field.variationPoint.short Label vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=40
majorVersion	PositiveInteger	0..1	attr	Major version of the service contract. **Tags:** xml.sequenceOffset=10

▽

△

Class	ServiceInterface			
method	ClientServerOperation	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=method.shortName, method.variation Point.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=50
minorVersion	PositiveInteger	0..1	attr	Minor version of the service contract. **Tags:** xml.sequenceOffset=20
trigger	Trigger	*	aggr	This represents the collection of triggers defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=trigger.shortName, trigger.variation Point.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=60

Table A.100: ServiceInterface

Class	*ServiceInterfaceDeployment* (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	Middleware transport layer specific configuration settings for the ServiceInterface and all contained ServiceInterface elements.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Subclasses	DdsServiceInterfaceDeployment, SomeipServiceInterfaceDeployment, UserDefinedServiceInterface Deployment			
Aggregated by	ARPackage.element			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
event Deployment	ServiceEvent Deployment	*	aggr	Middleware transport layer specific configuration settings for an Event that is defined in the ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=eventDeployment.shortName, event Deployment.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime
fieldDeployment	ServiceField Deployment	*	aggr	Middleware transport layer specific configuration settings for a Field that is defined in the ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=fieldDeployment.shortName, field Deployment.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime
method Deployment	ServiceMethod Deployment	*	aggr	Middleware transport layer specific configuration settings for a method that is defined in the ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=methodDeployment.shortName, method Deployment.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime

▽

△

Class	ServiceInterfaceDeployment (abstract)			
serviceInterface	ServiceInterface	0..1	ref	Reference to a ServiceInterface that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef

Table A.101: ServiceInterfaceDeployment

Class	ServiceInterfaceElementSecureComConfig			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	This element allows to secure the communication of the referenced ServiceInterface element.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AdaptivePlatformServiceInstance*.secureComConfig			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataId	PositiveInteger	0..1	attr	This attribute defines a unique numerical identifier for the referenced ServiceInterface element.
event	ServiceEvent Deployment	0..1	ref	Reference to an event that is protected by a security protocol.
fieldNotifier	ServiceField Deployment	0..1	ref	Reference to a field notifier that is protected by a security protocol.
freshnessValue Id	PositiveInteger	0..1	attr	This attribute defines the Id of the Freshness Value.
getterCall	ServiceField Deployment	0..1	ref	Reference to a field getter call message that is protected by a security protocol.
getterReturn	ServiceField Deployment	0..1	ref	Reference to a field getter return message that is protected by a security protocol.
methodCall	ServiceMethod Deployment	0..1	ref	Reference to a method call message that is protected by a security protocol.
methodReturn	ServiceMethod Deployment	0..1	ref	Reference to a method return message that is protected by a security protocol.
securedRx Verification	Boolean	0..1	attr	This attribute defines whether the ServiceInterface element shall verify its security credentials during reception.
setterCall	ServiceField Deployment	0..1	ref	Reference to a field setter call message that is protected by a security protocol.
setterReturn	ServiceField Deployment	0..1	ref	Reference to a field setter return message that is protected by a security protocol.

Table A.102: ServiceInterfaceElementSecureComConfig

Class	ServiceMethodDeployment (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	This abstract meta-class represents the ability to specify a deployment of a Method to a middleware transport layer.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	SomeipMethodDeployment, UserDefinedMethodDeployment			
Aggregated by	*ServiceInterfaceDeployment*.methodDeployment			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	***ServiceMethodDeployment*** (abstract)			
method	ClientServerOperation	0..1	ref	Reference to a method that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef

Table A.103: ServiceMethodDeployment

Enumeration	**ServiceVersionAcceptanceKindEnum**
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances
Note	Defined the possible acceptance kinds for required service instances.
Aggregated by	ConsumedServiceInstance.versionDrivenFindBehavior, RequiredSomeipServiceInstance.version DrivenFindBehavior
Literal	*Description*
exactOrAnyMinor Version	Search for ANY or specific minor version service instance and select either ALL returned service instances (in case of ANY) or exactly the specific minor version service instances defined in required MinorVersion. **Tags:** atp.EnumerationLiteralIndex=0
minimumMinor Version	Search for ANY minor version service instance and select only those service instances which have an equal or greater minor version than given in requiredMinorVersion. **Tags:** atp.EnumerationLiteralIndex=1

Table A.104: ServiceVersionAcceptanceKindEnum

Class	**SignalBasedEventElementToISignalTriggeringMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SignalBasedCommunication			
Note	This meta-class defines the mapping of a ServiceInterface event or an element that is defined inside of the event in case that the datatype is composite to an ISignalTriggering. **Tags:** atp.Status=candidate			
Base	*ARObject*, *AbstractSignalBasedToISignalTriggeringMapping*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ServiceInstanceToSignalMapping.eventElementMapping			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
dataPrototypeIn ServiceInterface Ref	DataPrototypeInService InterfaceRef	0..1	aggr	Reference to a DataPrototype or to an internal structure of a DataPrototype in the context of a ServiceInterface. **Tags:** atp.Status=candidate
filter	DataFilter	0..1	aggr	Defines an optional filter to be applied during translation. **Tags:** atp.Status=candidate
iSignal Triggering	ISignalTriggering	0..1	ref	Reference to the ISignalTriggering that is used to transport a piece of data of an event that is defined in a ServiceInterface in a signal-based way over a communication channel. **Tags:** atp.Status=candidate
transmission Trigger	Boolean	0..1	attr	Defines whether the source element triggers the sending of the respective payload. **Tags:** atp.Status=candidate

Table A.105: SignalBasedEventElementToISignalTriggeringMapping

Class	SomeipCollectionProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	Collection of attributes that are configurable for an event that is provided by a ServiceInstance or for a method that is provided or requested by a ServiceInstance.			
Base	*ARObject*			
Aggregated by	SomeipEventProps.collectionProps, SomeipMethodProps.collectionProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
udpCollection BufferTimeout	TimeValue	0..1	attr	Maximum time, an outgoing message (event, method call or method response) may be delayed, due to data collection.
udpCollection Trigger	UdpCollectionTrigger Enum	0..1	attr	Defines whether the ServiceInterface element (event or method) contributes to the triggering of the udp data transmission if data collection is enabled.

Table A.106: SomeipCollectionProps

Class	SomeipDataPrototypeTransformationProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::SerializationProperties			
Note	This meta-class represents the ability to define data transformation props specifically for a SOME/IP serialization for a given DataPrototype. **Tags:** atp.recommendedPackage=SomeipDataPrototypeTransformationPropss			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataPrototype	DataPrototypeInService InterfaceRef	*	aggr	Collection of DataPrototypes for which the settings in SomeipDataPrototypeTransformationProps are valid. For reuse reasons the SomeipDataPrototypeTransformation Props is able to aggregate several DataPrototypes.
network Representation	SwDataDefProps	0..1	aggr	Optional specification of the actual network representation for the referenced primitive DataPrototype. If a network representation is provided then the baseType available in the SwDataDefProps shall be used as input for the serialization/deserialization. If the network Representation is not provided then the baseType of the AbstractImplementationDataType shall be used for the serialization/deserialization. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=networkRepresentation
someip Transformation Props	ApSomeip TransformationProps	0..1	ref	This reference represents the ability to define data transformation props specifically for a SOME/IP serialization.

Table A.107: SomeipDataPrototypeTransformationProps

Class	SomeipEventDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for an Event.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceEventDeployment*			
Aggregated by	*ServiceInterfaceDeployment*.eventDeployment, SomeipFieldDeployment.notifier			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**SomeipEventDeployment**			
burstSize	PositiveInteger	0..1	attr	Specifies the number of segments that shall be transmitted in a burst ignoring separationTime. SeparationTime will then only be applied between bursts. If not configured, SeparationTime will be applied between all frames.
eventId	PositiveInteger	0..1	attr	Unique Identifier within a ServiceInterface that identifies the Event in SOME/IP. This Identifier is sent as part of the Message ID in SOME/IP messages.
eventReception DefaultValue	ValueSpecification	0..1	aggr	Value used to fill the Event data on the receiver side, if less data than expected is received. The value is expected to cover the entire expected event network payload. The value specification is supposed to take the order of serialized representation of the data on the network, as opposed to the order of elements in a data type description.
maximum SegmentLength	PositiveInteger	0..1	attr	This attribute describes the length in bytes of the SOME/IP segment. This includes 8 bytes for the Request ID, Protocol Version, Interface Version, Message Type and Return Code and 4 additional SOME/IP TP bytes. If this attribute is set to a value and the data length is larger than maximumSegmentLength then the corresponding SOME/IP message will be segmented into smaller parts that are transmitted over the network.
segment Reception TimeoutTime	TimeValue	0..1	attr	Timer to monitor the successful reception of segments (in seconds) in SOME/IP.
separationTime	TimeValue	0..1	attr	Sets the duration of the minimum time in seconds SOME/IP shall wait between the transmissions of segments.
serializer	SerializationTechnology Enum	0..1	attr	Defines which serialization technology shall be used.
transport Protocol	TransportLayerProtocol Enum	0..1	attr	This attribute defines over which Transport Layer Protocol this event is intended to be sent.

Table A.108: SomeipEventDeployment

Class	**SomeipEventGroup**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	Grouping of events and notification events inside a ServiceInterface in order to allow subscriptions.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	SomeipServiceInterfaceDeployment.eventGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
event	SomeipEvent Deployment	*	ref	Reference to an event that is part of the EventGroup.
eventGroupId	PositiveInteger	0..1	attr	Unique Identifier that identifies the EventGroup in SOME/IP. This Identifier is sent as Eventgroup ID in SOME/IP Service Discovery messages.

Table A.109: SomeipEventGroup

Class	SomeipEventProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class allows to set configuration options for an event in the provided service instance.			
Base	*ARObject*			
Aggregated by	ProvidedSomeipServiceInstance.eventProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
collectionProps	SomeipCollectionProps	0..1	aggr	Collection of timing attributes configurable for an event that is provided by a Service Instance.
event	SomeipEvent Deployment	0..1	ref	Reference to the event for which the SomeipEventProps are applicable.

Table A.110: SomeipEventProps

Class	SomeipFieldDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a Field.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceFieldDeployment*			
Aggregated by	*ServiceInterfaceDeployment*.fieldDeployment			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
get	SomeipMethod Deployment	0..1	aggr	This aggregation represents the setting of the get method.
notifier	SomeipEvent Deployment	0..1	aggr	This aggregation represents the settings of the notifier.
set	SomeipMethod Deployment	0..1	aggr	This aggregation represents the settings of the set method

Table A.111: SomeipFieldDeployment

Class	SomeipMethodDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a Method.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceMethodDeployment*			
Aggregated by	*ServiceInterfaceDeployment*.methodDeployment, SomeipFieldDeployment.get, SomeipFieldDeployment.set			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
burstSize Request	PositiveInteger	0..1	attr	Specifies the number of segments for the Method Call that shall be transmitted in a burst ignoring separation Time. SeparationTime will then only be applied between bursts. If not configured, SeparationTime will be applied between all frames.
burstSize Response	PositiveInteger	0..1	attr	Specifies the number of segments for the Method Response that shall be transmitted in a burst ignoring separationTime. SeparationTime will then only be applied between bursts. If not configured, SeparationTime will be applied between all frames.

▽

△

Class	**SomeipMethodDeployment**			
maximum SegmentLength Request	PositiveInteger	0..1	attr	This attribute describes the length in bytes of one SOME/IP segment into which the Method Call Message will be divided. This length field includes 8 bytes for the Request ID, Protocol Version, Interface Version, Message Type and Return Code and 4 additional SOME/IP TP bytes. If this attribute is set to a value and the data length is larger than maximumSegmentLengthRequest then the corresponding SOME/IP message will be segmented into smaller parts that are transmitted over the network.
maximum SegmentLength Response	PositiveInteger	0..1	attr	This attribute describes the length in bytes of one SOME/IP segment into which the Method Return Message will be divided. This length field includes 8 bytes for the Request ID, Protocol Version, Interface Version, Message Type and Return Code and 4 additional SOME/IP TP bytes. If this attribute is set to a value and the data length is larger than maximumSegmentLengthResponse then the corresponding SOME/IP message will be segmented into smaller parts that are transmitted over the network.
methodId	PositiveInteger	0..1	attr	Unique Identifier within a ServiceInterface that identifies the Method in SOME/IP. This Identifier is sent as part of the Message ID in SOME/IP messages.
segment Reception TimeoutTime Request	TimeValue	0..1	attr	Timer to monitor the successful reception of segments (in seconds) in SOME/IP for the Method Call.
segment Reception TimeoutTime Response	TimeValue	0..1	attr	Timer to monitor the successful reception of segments (in seconds) in SOME/IP for the Method Response.
separationTime Request	TimeValue	0..1	attr	Sets the duration of the minimum time in seconds SOME/IP shall wait between the transmissions of segments into which the Method Call Message will be divided.
separationTime Response	TimeValue	0..1	attr	Sets the duration of the minimum time in seconds SOME/IP shall wait between the transmissions of segments into which the Method Return Message will be divided.
transport Protocol	TransportLayerProtocol Enum	0..1	attr	This attribute defines over which Transport Layer Protocol this method is intended to be sent.

Table A.112: SomeipMethodDeployment

Class	**SomeipMethodProps**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	This meta-class allows to set configuration options for a method in the service instance.			
Base	*ARObject*			
Aggregated by	ProvidedSomeipServiceInstance.methodResponseProps, RequiredSomeipServiceInstance.method RequestProps			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
collectionProps	SomeipCollectionProps	0..1	aggr	Collection of timing attributes configurable for a method that is provided or requested by a Service Instance.
method	SomeipMethod Deployment	0..1	ref	Reference to the method for which the SomeipMethod Props are applicable.

Table A.113: SomeipMethodProps

Class	**SomeipProvidedEventGroup**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	The meta-class represents the ability to configure ServiceInstance related communication settings on the provided side for each EventGroup separately.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	ProvidedSomeipServiceInstance.providedEventGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
eventGroup	SomeipEventGroup	0..1	ref	Reference to the SomeipEventGroup in the System Manifest for which the ServiceInstance related Event Group settings are valid.
eventMulticast UdpPort	PositiveInteger	0..1	attr	UdpPort configuration that is used for Event communication in the IP-Multicast case. During SOME/IP Service Discovery: Send in the SD-SubscribeEventGroupAck Message to client (answer to SD-SubscribeEventGroup). Event: This is the destination-port where the server sends the multicast event messages if the multicastThreshold is exceeded.
ipv4MulticastIp Address	Ip4AddressString	0..1	attr	Multicast IPv4 Address that is transmitted in the Event GroupSubscribeAck message.
ipv6MulticastIp Address	Ip6AddressString	0..1	attr	Multicast IPv6 Address that is transmitted in the Event GroupSubscribeAck message.
multicast Threshold	PositiveInteger	0..1	attr	Specifies the number of subscribed clients that trigger the server to change the transmission of events to multicast. Example: If configured to 0 only unicast will be used. If configured to 1 the first client will be already served by multicast. If configured to 2 the first client will be served with unicast and as soon as the 2nd client arrives both will be served by multicast. This does not influence the handling of initial events, which are served using unicast only.
sdServerEvent GroupTiming Config	SomeipSdServerEvent GroupTimingConfig	0..1	ref	Server Timing configuration settings that are EventGroup specific.

Table A.114: SomeipProvidedEventGroup

Class	**SomeipRequiredEventGroup**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	The meta-class represents the ability to configure ServiceInstance related communication settings on the required side for each EventGroup separately.			
Base	*ARObject*, *Referrable*			
Aggregated by	RequiredSomeipServiceInstance.requiredEventGroup			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
eventGroup	SomeipEventGroup	0..1	ref	Reference to the SomeipEventGroup in the System Manifest for which the ServiceInstance related Event Group settings are valid.
sdClientEvent GroupTiming Config	SomeipSdClientEvent GroupTimingConfig	0..1	ref	Client Timing configuration settings that are EventGroup specific.

Table A.115: SomeipRequiredEventGroup

Class	SomeipSdClientEventGroupTimingConfig			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	This meta-class is used to specify configuration related to service discovery in the context of an event group on SOME/IP. **Tags:** atp.recommendedPackage=SomeipSdTimingConfigs			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
request ResponseDelay	RequestResponseDelay	0..1	aggr	The Service Discovery shall delay answers to unicast messages triggered by multicast messages (e.g. Subscribe Eventgroup after Offer Service).
subscribe Eventgroup RetryDelay	TimeValue	0..1	attr	This attribute defines the interval in seconds to re-trigger a subscription to a Eventgroup, if a retry to subscribe to a Eventgroup is configured (subscribeEventgroupRetryMax > 0).
subscribe Eventgroup RetryMax	PositiveInteger	0..1	attr	This attribute define the maximum counts of retries to subscribe to an Eventgroup. If the value is set to 0 no retry shall be done. If the value is set to 255 the retry shall be done as along as the Eventgroup is requested and no SubscribeEventGroupAck was received.
timeToLive	PositiveInteger	0..1	attr	Defines the time in seconds the subscription of this event is expected by the client. this value is sent from the client to the server in the SD-subscribeEvent message.

Table A.116: SomeipSdClientEventGroupTimingConfig

Class	SomeipSdClientServiceInstanceConfig			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	Client specific settings that are relevant for the configuration of SOME/IP Service-Discovery. **Tags:** atp.recommendedPackage=SomeipSdTimingConfigs			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
initialFind Behavior	InitialSdDelayConfig	0..1	aggr	Controls initial find behavior of clients.
priority	PositiveInteger	0..1	attr	This attribute defines the VLAN frame priority for Service Discovery messages that result from RequiredSomeip ServiceInstances that are referncing this SomeipSdClient ServiceInstanceConfig (Find, SubscribeEventGroup, Stop SubscribeEventgroup). Values from 0 (best effort) to 7 (highest) are allowed.

Table A.117: SomeipSdClientServiceInstanceConfig

Class	SomeipSdServerServiceInstanceConfig
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances
Note	Server specific settings that are relevant for the configuration of SOME/IP Service-Discovery. **Tags:** atp.recommendedPackage=SomeipSdTimingConfigs
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*

▽

△

Class	**SomeipSdServerServiceInstanceConfig**			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initialOffer Behavior	InitialSdDelayConfig	0..1	aggr	Controls offer behavior of the server.
offerCyclicDelay	TimeValue	0..1	attr	Optional attribute to define cyclic offers. Cyclic offer is active, if the delay is set (in seconds) and greater then 0.
priority	PositiveInteger	0..1	attr	This attribute defines the VLAN frame priority for Service Discovery messages that result from ProvidedSomeip ServiceInstances that are referencing the SomeipSd ServerServiceInstanceConfig (OfferService, StopOffer Service, SubscribeEventGroupAck). Values from 0 (best effort) to 7 (highest) are allowed.
request ResponseDelay	RequestResponseDelay	0..1	aggr	Maximum/Minimum allowable response delay to entries received by multicast in seconds. The Service Discovery shall delay answers to entries that were transported in a multicast SOME/IP-SD message (e.g. FindService).
serviceOffer TimeToLive	PositiveInteger	0..1	attr	Defines the time in seconds the service offer is valid.

Table A.118: SomeipSdServerServiceInstanceConfig

Class	**SomeipServiceInstanceToMachineMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceMapping			
Note	This meta-class allows to map SomeipServiceInstances to a CommunicationConnector of a Machine. In this step the network configuration (IP Address, Transport Protocol, Port Number) for the ServiceInstance is defined. **Tags:** atp.recommendedPackage=ServiceInstanceToMachineMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *ServiceInstanceToMachineMapping*, *UploadableDesignElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
remoteMulticast Config	SomeipRemote MulticastConfig	*	ref	This reference defines a remote multicast Address (IP Address, Port) that is used in a static configuration to setup the communication path between a service provider and service consumer. This reference shall ONLY be used if the remote address is determined from the configuration and not at runtime from the Service Discovery. **Tags:** atp.Status=candidate
remoteUnicast Config	SomeipRemoteUnicast Config	*	ref	In case that a static service connection is used and a single peer exists this element is used to statically configure the remote peer's address. **Tags:** atp.Status=candidate
tcpPort	ApApplicationEndpoint	0..1	ref	local TcpPort that will be used by the ServiceInstance for the communication.
udpCollection BufferSize Threshold	PositiveInteger	0..1	attr	Specifies the amount of data in bytes that shall be buffered for data transmission over the udp connection specified by this SomeipServiceInstanceToMachine Mapping. If this attribute is set to a value, then the data collection feature is enabled.
udpPort	ApApplicationEndpoint	0..1	ref	local UdpPort that will be used by the ServiceInstance for the communication.

Table A.119: SomeipServiceInstanceToMachineMapping

Class	SomeipServiceInterfaceDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a ServiceInterface. **Tags:** atp.recommendedPackage=ServiceInterfaceDeployments			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *ServiceInterfaceDeployment*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
eventGroup	SomeipEventGroup	*	aggr	SOME/IP EventGroups that are defined within the SOME/IP ServiceClass.
serviceInterfaceId	PositiveInteger	0..1	attr	Unique Identifier that identifies the ServiceInterface in SOME/IP. This Identifier is sent as Service ID in SOME/IP Service Discovery messages.
serviceInterfaceVersion	SomeipServiceVersion	0..1	aggr	The SOME/IP major and minor Version of the Service.

Table A.120: SomeipServiceInterfaceDeployment

Class	SomeipServiceVersion			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	This meta-class represents the ability to describe a version of a SOME/IP Service.			
Base	*ARObject*			
Aggregated by	ConsumedServiceInstance.blocklistedVersion, RequiredSomeipServiceInstance.blocklistedVersion, SomeipServiceInterfaceDeployment.serviceInterfaceVersion			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
majorVersion	PositiveInteger	0..1	attr	Major Version of the ServiceInterface. **Tags:** xml.sequenceOffset=10
minorVersion	PositiveInteger	0..1	attr	Minor Version of the ServiceInterface. **Tags:** xml.sequenceOffset=20

Table A.121: SomeipServiceVersion

Class	StdCppImplementationDataType			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::CppImplementationDataType			
Note	This meta-class represents the way to specify a data type definition that is taken as the basis for a C++ language binding to a C++ Standard Library feature. **Tags:** atp.recommendedPackage=CppImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataType*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.122: StdCppImplementationDataType

Class	«atpVariation» **SwDataDefProps**
Package	M2::MSR::DataDictionary::DataDefProperties
Note	This class is a collection of properties relevant for data objects under various aspects. One could consider this class as a "pattern of inheritance by aggregation". The properties can be applied to all objects of all classes in which SwDataDefProps is aggregated. Note that not all of the attributes or associated elements are useful all of the time. Hence, the process definition (e.g. expressed with an OCL or a Document Control Instance MSR-DCI) has the task of implementing limitations. SwDataDefProps covers various aspects: • Structure of the data element for calibration use cases: is it a single value, a curve, or a map, but also the recordLayouts which specify how such elements are mapped/converted to the DataTypes in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, swAddr Method, swPointerTagetProps, baseType, implementationDataType and additionalNativeTypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalid Value • Code generation policy provided by swRecordLayout **Tags:** vh.latestBindingTime=codeGenerationTime
Base	*ARObject*
Aggregated by	*AutosarDataType*.swDataDefProps, CompositeNetworkRepresentation.networkRepresentation, CppImplementationDataTypeElement.swDataDefProps, *DataPrototype*.swDataDefProps, DataPrototypeTransformationProps.networkRepresentationProps, DiagnosticDataElement.swDataDefProps, DiagnosticEnvDataElementCondition.swDataDefProps, DltArgument.networkRepresentation, FlatInstanceDescriptor.swDataDefProps, ImplementationDataTypeElement.swDataDefProps, InstantiationDataDefProps.swDataDefProps, ISignal.networkRepresentationProps, McDataInstance.resultingProperties, ParameterAccess.swDataDefProps, PerInstanceMemory.swDataDefProps, *ReceiverComSpec*.networkRepresentation, SecurityEventContextDataElement.networkRepresentation, *SenderComSpec*.networkRepresentation, SomeipDataPrototypeTransformationProps.networkRepresentation, SwPointerTargetProps.swDataDefProps, SwServiceArg.swDataDefProps, SwSystemconst.swDataDefProps, SystemSignal.physicalProps

Attribute	***Type***	***Mult.***	***Kind***	***Note***
additionalNativeTypeQualifier	NativeDeclarationString	0..1	attr	This attribute is used to declare native qualifiers of the programming language which can neither be deduced from the baseType (e.g. because the data object describes a pointer) nor from other more abstract attributes. Examples are qualifiers like "volatile", "strict" or "enum" of the C-language. All such declarations have to be put into one string. **Tags:** xml.sequenceOffset=235
annotation	Annotation	*	aggr	This aggregation allows to add annotations (yellow pads ...) related to the current data object. **Tags:** xml.roleElement=true xml.roleWrapperElement=true xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false
baseType	SwBaseType	0..1	ref	Base type associated with the containing data object. **Tags:** xml.sequenceOffset=50
compuMethod	CompuMethod	0..1	ref	Computation method associated with the semantics of this data object. **Tags:** xml.sequenceOffset=180

▽

△

Class	«atpVariation» **SwDataDefProps**			
dataConstr	DataConstr	0..1	ref	Data constraint for this data object. **Tags:** xml.sequenceOffset=190
displayFormat	DisplayFormatString	0..1	attr	This property describes how a number is to be rendered e.g. in documents or in a measurement and calibration system. **Tags:** xml.sequenceOffset=210
display Presentation	DisplayPresentation Enum	0..1	attr	This attribute controls the presentation of the related data for measurement and calibration tools.
implementation DataType	AbstractImplementation DataType	0..1	ref	This association denotes the ImplementationDataType of a data declaration via its aggregated SwDataDefProps. It is used whenever a data declaration is not directly referring to a base type. Especially • redefinition of an ImplementationDataType via a "typedef" to another ImplementationDatatype • the target type of a pointer (see SwPointerTarget Props), if it does not refer to a base type directly • the data type of an array or record element within an ImplementationDataType, if it does not refer to a base type directly • the data type of an SwServiceArg, if it does not refer to a base type directly **Tags:** xml.sequenceOffset=215
invalidValue	ValueSpecification	0..1	aggr	Optional value to express invalidity of the actual data element. **Tags:** xml.sequenceOffset=255
stepSize	Float	0..1	attr	This attribute can be used to define a value which is added to or subtracted from the value of a DataPrototype when using up/down keys while calibrating.
swAddrMethod	SwAddrMethod	0..1	ref	Addressing method related to this data object. Via an association to the same SwAddrMethod it can be specified that several DataPrototypes shall be located in the same memory without already specifying the memory section itself. **Tags:** xml.sequenceOffset=30
swAlignment	AlignmentType	0..1	attr	The attribute describes the intended typical alignment of the DataPrototype. If the attribute is not defined the alignment is determined by the swBaseType size and the memoryAllocationKeywordPolicy of the referenced Sw AddrMethod. **Tags:** xml.sequenceOffset=33
swBit Representation	SwBitRepresentation	0..1	aggr	Description of the binary representation in case of a bit variable. **Tags:** xml.sequenceOffset=60
swCalibration Access	SwCalibrationAccess Enum	0..1	attr	Specifies the read or write access by MCD tools for this data object. **Tags:** xml.sequenceOffset=70
swCalprmAxis Set	SwCalprmAxisSet	0..1	aggr	This specifies the properties of the axes in case of a curve or map etc. This is mainly applicable to calibration parameters. **Tags:** xml.sequenceOffset=90

▽

△

Class	«atpVariation» **SwDataDefProps**			
swComparison Variable	SwVariableRefProxy	*	aggr	Variables used for comparison in an MCD process. **Tags:** xml.sequenceOffset=170 xml.typeElement=false
swData Dependency	SwDataDependency	0..1	aggr	Describes how the value of the data object has to be calculated from the value of another data object (by the MCD system). **Tags:** xml.sequenceOffset=200
swHostVariable	SwVariableRefProxy	0..1	aggr	Contains a reference to a variable which serves as a host-variable for a bit variable. Only applicable to bit objects. **Tags:** xml.sequenceOffset=220 xml.typeElement=false
swImplPolicy	SwImplPolicyEnum	0..1	attr	Implementation policy for this data object. **Tags:** xml.sequenceOffset=230
swIntended Resolution	Numerical	0..1	attr	The purpose of this element is to describe the requested quantization of data objects early on in the design process. The resolution ultimately occurs via the conversion formula present (compuMethod), which specifies the transition from the physical world to the standardized world (and vice-versa) (here, "the slope per bit" is present implicitly in the conversion formula). In the case of a development phase without a fixed conversion formula, a pre-specification can occur through swIntendedResolution. The resolution is specified in the physical domain according to the property "unit". **Tags:** xml.sequenceOffset=240
swInterpolation Method	Identifier	0..1	attr	This is a keyword identifying the mathematical method to be applied for interpolation. The keyword needs to be related to the interpolation routine which needs to be invoked. **Tags:** xml.sequenceOffset=250
swIsVirtual	Boolean	0..1	attr	This element distinguishes virtual objects. Virtual objects do not appear in the memory, their derivation is much more dependent on other objects and hence they shall have a swDataDependency . **Tags:** xml.sequenceOffset=260
swPointerTarget Props	SwPointerTargetProps	0..1	aggr	Specifies that the containing data object is a pointer to another data object. **Tags:** xml.sequenceOffset=280
swRecord Layout	SwRecordLayout	0..1	ref	Record layout for this data object. **Tags:** xml.sequenceOffset=290

▽

△

Class	«atpVariation» **SwDataDefProps**			
swRefresh Timing	MultidimensionalTime	0..1	aggr	This element specifies the frequency in which the object involved shall be or is called or calculated. This timing can be collected from the task in which write access processes to the variable run. But this cannot be done by the MCD system. So this attribute can be used in an early phase to express the desired refresh timing and later on to specify the real refresh timing. **Tags:** xml.sequenceOffset=300
swTextProps	SwTextProps	0..1	aggr	the specific properties if the data object is a text object. **Tags:** xml.sequenceOffset=120
swValueBlock Size	Numerical	0..1	attr	This represents the size of a Value Block **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=80
swValueBlock SizeMult (ordered)	Numerical	*	attr	This attribute is used to specify the dimensions of a value block (VAL_BLK) for the case that that value block has more than one dimension. The dimensions given in this attribute are ordered such that the first entry represents the first dimension, the second entry represents the second dimension, and so on. For one-dimensional value blocks the attribute swValue BlockSize shall be used and this attribute shall not exist. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime
unit	Unit	0..1	ref	Physical unit associated with the semantics of this data object. This attribute applies if no compuMethod is specified. If both units (this as well as via compuMethod) are specified the units shall be compatible. **Tags:** xml.sequenceOffset=350
valueAxisData Type	ApplicationPrimitive DataType	0..1	ref	The referenced ApplicationPrimitiveDataType represents the primitive data type of the value axis within a compound primitive (e.g. curve, map). It supersedes CompuMethod, Unit, and BaseType. **Tags:** xml.sequenceOffset=355

Table A.123: SwDataDefProps

Class	**SwTextProps**			
Package	M2::MSR::DataDictionary::DataDefProperties			
Note	This meta-class expresses particular properties applicable to strings in variables or calibration parameters.			
Base	*ARObject*			
Aggregated by	SwDataDefProps.swTextProps			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	**SwTextProps**			
arraySize Semantics	ArraySizeSemantics Enum	0..1	attr	This attribute controls the semantics of the arraysize for the array representing the string in an Implementation DataType. It is there to support a safe conversion between ApplicationDatatype and ImplementationDatatype, even for variable length strings as required e.g. for Support of SAE J1939.
baseType	SwBaseType	0..1	ref	This is the base type of one character in the string. In particular this baseType denotes the intended encoding of the characters in the string on level of ApplicationData Type. **Tags:** xml.sequenceOffset=30
swFillCharacter	Integer	0..1	attr	Filler character for text parameter to pad up to the maximum length swMaxTextSize. The value will be interpreted according to the encoding specified in the associated base type of the data object, e.g. 0x30 (hex) represents the ASCII character zero as filler character and 0 (dec) represents an end of string as filler character. The usage of the fill character depends on the arraySize Semantics. **Tags:** xml.sequenceOffset=40
swMaxTextSize	Integer	0..1	attr	Specifies the maximum text size in characters. Note the size in bytes depends on the encoding in the corresponding baseType. **Stereotypes:** atpVariation **Tags:** vh.latestBindingTime=preCompileTime xml.sequenceOffset=20

Table A.124: SwTextProps

Class	**SymbolProps**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Components			
Note	This meta-class represents the ability to contribute a part of a namespace.			
Base	*ARObject*, *ImplementationProps*, *Referrable*			
Aggregated by	Allocator.namespace, ApApplicationErrorDomain.namespace, *AtomicSwComponentType*.symbolProps, *CppImplementationDataType*.namespace, ImplementationDataType.symbolProps, *PortInterface*.namespace, SecurityEventDefinition.eventSymbolName			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.125: SymbolProps

Primitive	**TimeValue**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive type is taken for expressing time values. The numerical value is supposed to be interpreted in the physical unit second. **Tags:** xml.xsd.customType=TIME-VALUE xml.xsd.type=double

Table A.126: TimeValue

Class	TlsIamRemoteSubject			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SCREIAM			
Note	This meta-class defines the proxy information about the remote node in case of TLS. **Tags:** atp.Status=candidate atp.recommendedPackage=IamRemoteSubjects			
Base	*ARElement*, *ARObject*, *AbstractIamRemoteSubject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
acceptedCryptoCipherSuiteWithPsk	TlsCryptoCipherSuite	*	ref	This reference is used to identify a remote node by means of the preshared Key. **Tags:** atp.Status=candidate
acceptedRemoteCertificate	CryptoServiceCertificate	*	ref	This reference is used to identify a remote node by means of the certificate. **Tags:** atp.Status=candidate
certCommonName	String	0..1	attr	This attribute defines the common name (CN) of the certificate of the remote peer. **Tags:** atp.Status=candidate
derivedCertificateAccepted	Boolean	0..1	attr	This attribute defines whether a derivedCertificate is accepted (true) or not (false). **Tags:** atp.Status=candidate
iamRelevantTlsSecureComProps	TlsSecureComProps	*	ref	This reference defines the local TlsSecureComProps that are relevant for IAM. **Tags:** atp.Status=candidate

Table A.127: TlsIamRemoteSubject

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*, *PackageableElement*, *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.128: TlsSecureComProps

Class	TlvDataIdDefinition
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer
Note	This meta-class represents the ability to define the tlvDataId.
Base	*ARObject*

▽

△

Class	**TlvDataIdDefinition**			
Aggregated by	TlvDataIdDefinitionSet.tlvDataIdDefinition			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
id	PositiveInteger	0..1	attr	This attribute represents the definition of the value of the TlvDataId **Stereotypes:** atpIdentityContributor
tlvArgument	ArgumentDataPrototype	0..1	ref	This reference assigns a tlvDataId to a given argument of a ClientServerOperation.
tlv ImplementationDataType Element	AbstractImplementationDataTypeElement	0..1	ref	This reference associates the definition of a TLV data id with a given AbstractImplementationDataTypeElement.
tlvRecord Element	ApplicationRecord Element	0..1	ref	This reference associates the definition of a TLV data id with a given ApplicationRecordElement.

Table A.129: TlvDataIdDefinition

Class	**TransformationPropsToServiceInterfaceElementMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure			
Note	This meta-class represents the ability to associate a ServiceInterface element with TransformationProps. The referenced elements of the Service Interface will be serialized according to the settings defined in the TransformationProps. **Tags:** atp.recommendedPackage=TransformationPropsToServiceInterfaceElementMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
event	VariableDataPrototype	*	ref	This represents the reference to one or several events of one ServiceInterface.
field	Field	*	ref	This represents the reference to one or several fields of one ServiceInterface.
methodCall	ClientServerOperation	*	ref	This represents the reference to one or several method calls of one ServiceInterface.
methodReturn	ClientServerOperation	*	ref	This represents the reference to one or several method return of one ServiceInterface.
tlvDataId Definition	TlvDataIdDefinitionSet	*	ref	This reference identifies the TlvDataIdDefinitions relevant for the enclosing TransformationPropsToServiceInterface Mapping.
transformation Props	TransformationProps	0..1	ref	This represents the reference to the applicable Serialization properties.
trigger	Trigger	*	ref	This represents the reference to one or several triggers of one ServiceInterface.

Table A.130: TransformationPropsToServiceInterfaceElementMapping

Enumeration	**TransportLayerProtocolEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment
Note	This enumeration allows to choose a TCP/IP transport layer protocol.
Aggregated by	SomeipEventDeployment.transportProtocol, SomeipMethodDeployment.transportProtocol
Literal	***Description***

▽

△

Enumeration	**TransportLayerProtocolEnum**
tcp	Transmission control protocol **Tags:** atp.EnumerationLiteralIndex=1
udp	User datagram protocol **Tags:** atp.EnumerationLiteralIndex=0

Table A.131: TransportLayerProtocolEnum

Class	**Trigger**			
Package	M2::AUTOSARTemplates::CommonStructure::TriggerDeclaration			
Note	The Trigger represents a special kind of an event (without data) at which occurrence the Service Consumer shall react in a particular manner.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, BswModuleDescription.releasedTrigger, BswModuleDescription.requiredTrigger, ServiceInterface.trigger, TriggerInterface.trigger			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.132: Trigger

Enumeration	**UdpCollectionTriggerEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment
Note	Defines whether the ServiceInterface element (event or method) contributes to the triggering of the udp data transmission if data collection is enabled.
Aggregated by	SomeipCollectionProps.udpCollectionTrigger
Literal	*Description*
always	ServiceInterface element will trigger the transmission of the data. **Tags:** atp.EnumerationLiteralIndex=0
never	ServiceInterface element will be buffered and will not trigger the transmission of the data. **Tags:** atp.EnumerationLiteralIndex=1

Table A.133: UdpCollectionTriggerEnum

Class	**UserDefinedEventDeployment**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	UserDefined configuration settings for an Event.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceEventDeployment*			
Aggregated by	*ServiceInterfaceDeployment*.eventDeployment, UserDefinedFieldDeployment.notifier			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	UserDefinedEventDeployment			
eventReception DefaultValue	ValueSpecification	0..1	aggr	Value used to fill the Event data on the receiver side, if less data than expected is received. The value is expected to cover the entire expected event network payload. The value specification is supposed to take the order of serialized representation of the data on the network, as opposed to the order of elements in a data type description.

Table A.134: UserDefinedEventDeployment

Class	UserDefinedServiceInstanceToMachineMapping			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceMapping			
Note	This meta-class allows to map UserDefinedServiceInstances to a CommunicationConnector of a Machine. **Tags:** atp.recommendedPackage=ServiceInstanceToMachineMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *ServiceInstanceToMachineMapping*, *UploadableDesignElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.135: UserDefinedServiceInstanceToMachineMapping

Class	UserDefinedServiceInterfaceDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	UserDefined configuration settings for a ServiceInterface. **Tags:** atp.recommendedPackage=ServiceInterfaceDeployments			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *ServiceInterfaceDeployment*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.136: UserDefinedServiceInterfaceDeployment

Class	VariableDataPrototype
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes
Note	A VariableDataPrototype represents a formalized generic piece of information that is typically mutable by the application software layer. VariableDataPrototype is used in various contexts and the specific context gives the otherwise generic VariableDataPrototype a dedicated semantics.
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*

▽

△

Class	**VariableDataPrototype**			
Aggregated by	ApplicationInterface.indication, *AtpClassifier*.atpFeature, BswInternalBehavior.arTypedPerInstance Memory, BswModuleDescription.providedData, BswModuleDescription.requiredData, BulkNvData Descriptor.bulkNvBlock, *InternalBehavior*.staticMemory, NvBlockDescriptor.ramBlock, NvDataInterface. nvData, SenderReceiverInterface.dataElement, ServiceInterface.event, SwcInternalBehavior.arTypedPer InstanceMemory, SwcInternalBehavior.explicitInterRunnableVariable, SwcInternalBehavior.implicitInter RunnableVariable			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table A.137: VariableDataPrototype

B Demands and constraints on Base Software (normative)

This functional cluster defines no demands or constraints for the Base Software on which the AUTOSAR Adaptive Platform is running on (usually a POSIX-compatible operating system).

C Platform Extension Interfaces (normative)

This chapter provides a reference of the Platform Extension Interfaces defined by this functional cluster. Platform Extension Interfaces are intended to be used/provided by an OEM or Integrator to extend the functionality of the AUTOSAR Adaptive Platform.

C.1 Header: apext/com/secoc/fvm.h

C.1.1 Struct: FVContainer

[SWS_CM_11286] Definition of API class apext::com::secoc::FVContainer

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	struct
Header file:	#include "apext/com/secoc/fvm.h"
Forwarding header file:	#include "apext/com/com_fwd.h"
Scope:	`namespace apext::com::secoc`
Symbol:	FVContainer
Syntax:	`struct FVContainer {...};`
Description:	A container to hold the length of freshness value in bits and the freshness value itself

⌋

C.1.1.1 Public Member Variables

C.1.1.1.1 length

[SWS_CM_11344] Definition of API variable apext::com::secoc::FVContainer::length

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	variable
Header file:	#include "apext/com/secoc/fvm.h"
Scope:	`struct apext::com::secoc::FVContainer`
Symbol:	length
Type:	`std::uint64_t`
Syntax:	`std::uint64_t length;`
Description:	Length in bits of the freshness value passed in `apext::com::secoc::FVContainer`

⌋

C.1.1.1.2 value

[SWS_CM_11345] Definition of API variable apext::com::secoc::FVContainer::value

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	variable
Header file:	#include "apext/com/secoc/fvm.h"
Scope:	`struct apext::com::secoc::FVContainer`
Symbol:	value
Type:	`ara::core::Vector< std::uint8_t >`
Syntax:	`ara::core::Vector<std::uint8_t> value;`
Description:	Vector of bytes containing the freshness value. Depending on whether the container is used as an input or returning value by the method it will contain either the full freshness or truncated values

⌋

C.1.2 Class: FVM

[SWS_CM_11287] Definition of API class apext::com::secoc::FVM

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	class
Header file:	#include "apext/com/secoc/fvm.h"
Forwarding header file:	#include "apext/com/com_fwd.h"
Scope:	`namespace apext::com::secoc`
Symbol:	FVM
Syntax:	`class FVM {...};`
Description:	A freshness value management interface to be implemented by the OEM/stack vendor. To be used by the freshness value management library implementer either OEM or stack vendor. The class will have a single instance in the CM.

⌋

C.1.2.1 Public Member Types

C.1.2.1.1 Enumeration: VerificationStatus

[SWS_CM_11378] Definition of API enum apext::com::secoc::FVM::VerificationStatus

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	enumeration	
Header file:	#include "apext/com/secoc/fvm.h"	
Forwarding header file:	#include "apext/com/com_fwd.h"	
Scope:	`class apext::com::secoc::FVM`	
Symbol:	VerificationStatus	
Underlying type:	std::uint8_t	
Syntax:	`enum class VerificationStatus : std::uint8_t {...};`	
Values:	kSecOcVerification Success= 0x00	Verification successful
	kSecOcVerification Failure= 0x01	Verification not successful
	kSecOcFreshness Failure= 0x02	Verification not successful because of wrong freshness value
	kSecOcAuthentication BuildFailure= 0x03	Verification not successful because of wrong build authentication codes
	kSecOcNoVerification= 0x04	Verification has been skipped and the data has been provided to the application as is
	kSecOcVerification FailureOverwritten= 0x05	Verification failed, but the PDU was passed on to the application due to the override status for this PDU
Description:	Defines the status of a verification	

⌋

C.1.2.2 Public Member Functions

C.1.2.2.1 Member Functions

C.1.2.2.1.1 GetRxFreshness

[SWS_CM_11288] Definition of API function apext::com::secoc::FVM::GetRx Freshness

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	function	
Header file:	#include "apext/com/secoc/fvm.h"	
Scope:	`class apext::com::secoc::FVM`	
Syntax:	`ara::core::Result< FVContainer > GetRxFreshness (std::uint16_t SecOCFreshnessValueID, const FVContainer &SecOCTruncatedFreshnessValue, std::uint16_t SecOCAuthVerifyAttempts) noexcept;`	
Template param:	FVContainer	As per [SWS_CM_11286]
Parameters (in):	SecOCFreshnessValue ID	the identifier of the freshness value.
	SecOCTruncated FreshnessValue	the `apext::com::secoc::FVContainer` with the values from the received Secured I-PDU/ message
	SecOCAuthVerify Attempts	the number of authentication verify attempts of this I-PDU/message since the last reception. The value is `0` for the first attempt and incremented on every unsuccessful verification attempt.
Return value:	ara::core::Result< FVContainer >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `apext::com::secoc::FVContainer` that holds the freshness value to be used for the calculation of the the authenticator by the SecOC or recoverable error. • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::com::ComErrc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComSecOcFvmErrc::kFVNotAvailable	rollback_semantics
		Freshness Value not available
Description:	Used by the SecOC to obtain the current freshness value.	

⌋

C.1.2.2.1.2 GetTxFreshness

[SWS_CM_11289] Definition of API function apext::com::secoc::FVM::GetTx Freshness

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	function	
Header file:	#include "apext/com/secoc/fvm.h"	
Scope:	`class apext::com::secoc::FVM`	
Syntax:	`ara::core::Result< FVContainer > GetTxFreshness (std::uint16_t SecOCFreshnessValueID) noexcept;`	
Template param:	FVContainer	As per [SWS_CM_11286]
Parameters (in):	SecOCFreshnessValueID	the identifier of the freshness value.
Return value:	ara::core::Result< FVContainer >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `apext::com::secoc::FVContainer` that holds the freshness value to be used for the calculation of the the authenticator by the SecOC or recoverable error. • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::com::ComErrc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComSecOcFvmErrc::kFVNotAvailable	rollback_semantics Freshness Value not available
Description:	Used by the SecOC to obtain the current freshness value	

⌋

C.1.2.2.1.3 Initialize

[SWS_CM_11290] Definition of API function apext::com::secoc::FVM::Initialize

Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	function
Header file:	#include "apext/com/secoc/fvm.h"
Scope:	`class apext::com::secoc::FVM`
Syntax:	`ara::core::Result< void > Initialize () noexcept;`

▽

△

Return value:	ara::core::Result< void >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `void` • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::com::ComErrc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Errors:	ComSecOcFvmErrc::kFVInitializeFailed	rollback_semantics
		Initialization of Freshness Value Manager failed
Description:	Initializes FVM plugin implementation	

⌋

C.1.2.2.1.4 SetVerificationStatus

[SWS_CM_19999] Definition of API function apext::com::secoc::FVM::SetVerificationStatus

Status: DRAFT
Upstream requirements: RS_CM_00801, RS_CM_00802, RS_CM_00803, RS_CM_00804

⌈

Kind:	function	
Header file:	#include "apext/com/secoc/fvm.h"	
Scope:	`class apext::com::secoc::FVM`	
Syntax:	`ara::core::Result< void > SetVerificationStatus (std::uint16_t secOCFreshnessValueID, VerificationStatus secOCVerificationStatus) noexcept;`	
Parameters (in):	secOCFreshnessValueID	the identifier of the freshness value.
	secOCVerificationStatus	verification status result for the given FreshnessValueID
Return value:	ara::core::Result< void >	• If successful: an `ara::core::Result` containing a `ara::core::Result::value_type` containing a `void` • If unsuccessful: an `ara::core::Result` containing an `ara::core::Result::error_type` i.e. a corresponding `ara::com::ComErrc`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	This method is used by the SecOC to report all verification statuses to the Freshness Value Manager.	

⌋

C.2 Header: apext/com/secoc/fvm_error_domain.h

C.2.1 Non-Member Types

C.2.1.1 Enumeration: ComSecOcFvmErrc

[SWS_CM_11342] Definition of API enum apext::com::secoc::ComSecOcFvmErrc

Upstream requirements: RS_AP_00130

⌈

Kind:	enumeration	
Header file:	#include "apext/com/secoc/fvm_error_domain.h"	
Forwarding header file:	#include "apext/com/com_fwd.h"	
Scope:	`namespace apext::com::secoc`	
Symbol:	ComSecOcFvmErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class ComSecOcFvmErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kFVNotAvailable= 1	Freshness Value not available
	kFVInitializeFailed= 2	Initialization of Freshness Value Manager failed
Description:	Defines the error codes for the `apext::com::secoc::ComSecOcFvmErrorDomain`	

⌋

C.2.2 Non-Member Functions

C.2.2.1 Other

C.2.2.1.1 GetComSecOcFvmErrorDomain

[SWS_CM_12521] Definition of API function apext::com::secoc::GetComSecOcFvmErrorDomain

Upstream requirements: RS_AP_00120, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "apext/com/secoc/fvm_error_domain.h"	
Scope:	`namespace apext::com::secoc`	
Syntax:	`constexpr const ara::core::ErrorDomain & GetComSecOcFvmErrorDomain () noexcept;`	
Return value:	const ara::core::ErrorDomain &	Reference to the `apext::com::secoc::ComSecOcFvmErrorDomain` object

▽

△

Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Returns a reference to the `apext::com::secoc::ComSecOcFvmErrorDomain` object

⌋

C.2.2.1.2 MakeErrorCode

[SWS_CM_12522] Definition of API function apext::com::secoc::MakeErrorCode

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "apext/com/secoc/fvm_error_domain.h"	
Scope:	`namespace apext::com::secoc`	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (apext::com::secoc::ComSecOcFvmErrc errorCode, ara::core::ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	errorCode	Error code number.
	data	Vendor defined data associated with the error
Return value:	ara::core::ErrorCode	An `ara::core::ErrorCode` object.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Creates an instance of `ara::core::ErrorCode`	

⌋

C.2.3 Class: ComSecOcFvmErrorDomain

[SWS_CM_12514] Definition of API class apext::com::secoc::ComSecOcFvmErrorDomain

Upstream requirements: RS_AP_00122, RS_AP_00127, RS_AP_00130

⌈

Kind:	class
Header file:	#include "apext/com/secoc/fvm_error_domain.h"
Forwarding header file:	#include "apext/com/com_fwd.h"
Scope:	`namespace apext::com::secoc`
Symbol:	ComSecOcFvmErrorDomain

▽

△

Base class:	ara::core::ErrorDomain
Syntax:	`class ComSecOcFvmErrorDomain final : public ara::core::ErrorDomain {...};`
Unique ID:	As per `ara::com::ComSecOcFvmErrorDomain` in [SWS_CORE_90023]
Description:	Defines a class representing the SecOc Freshness Value Manager error domain

⌋

C.2.3.1 Public Member Types

C.2.3.1.1 Type Alias: Errc

[SWS_CM_12515] Definition of API type apext::com::secoc::ComSecOcFvmErrorDomain::Errc

Upstream requirements: RS_AP_00120, RS_AP_00127, RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "apext/com/secoc/fvm_error_domain.h"
Scope:	`class apext::com::secoc::ComSecOcFvmErrorDomain`
Symbol:	Errc
Syntax:	`using Errc = ComSecOcFvmErrc;`
Description:	Alias for the error code value enumeration

⌋

C.2.3.1.2 Type Alias: Exception

[SWS_CM_12516] Definition of API type apext::com::secoc::ComSecOcFvmErrorDomain::Exception

Upstream requirements: RS_AP_00120, RS_AP_00127, RS_AP_00130

⌈

Kind:	type alias
Header file:	#include "apext/com/secoc/fvm_error_domain.h"
Scope:	`class apext::com::secoc::ComSecOcFvmErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = ComSecOcFvmException;`

▽

△

Description:	Alias for the exception base class

⌋

C.2.3.2 Public Member Functions

C.2.3.2.1 Special Member Functions

C.2.3.2.1.1 Default Constructor

[SWS_CM_12517] Definition of API function apext::com::secoc::ComSecOcFvmErrorDomain::ComSecOcFvmErrorDomain

Upstream requirements: RS_AP_00120, RS_AP_00127, RS_AP_00130

⌈

Kind:	function
Header file:	#include "apext/com/secoc/fvm_error_domain.h"
Scope:	`class apext::com::secoc::ComSecOcFvmErrorDomain`
Syntax:	`constexpr ComSecOcFvmErrorDomain () noexcept;`
Exception Safety:	exception safe
Thread Safety:	implementation defined
Description:	Constructs a new `apext::com::secoc::ComSecOcFvmErrorDomain` object

⌋

C.2.3.2.2 Member Functions

C.2.3.2.2.1 Message

[SWS_CM_12519] Definition of API function apext::com::secoc::ComSecOcFvmErrorDomain::Message

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function
Header file:	#include "apext/com/secoc/fvm_error_domain.h"
Scope:	`class apext::com::secoc::ComSecOcFvmErrorDomain`
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 `errorCode`
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns the message associated with `errorCode`	

⌋

C.2.3.2.2.2 Name

[SWS_CM_12518] Definition of API function apext::com::secoc::ComSecOcFvmErrorDomain::Name

Upstream requirements: RS_AP_00120, RS_AP_00127, RS_AP_00130

⌈

Kind:	function	
Header file:	#include "apext/com/secoc/fvm_error_domain.h"	
Scope:	`class apext::com::secoc::ComSecOcFvmErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"SecOcFvm"
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Returns a string constant associated with the `apext::com::secoc::ComSecOcFvmErrorDomain`	

⌋

C.2.3.2.2.3 ThrowAsException

[SWS_CM_12520] Definition of API function apext::com::secoc::ComSecOcFvmErrorDomain::ThrowAsException

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130

⌈

Kind:	function
Header file:	#include "apext/com/secoc/fvm_error_domain.h"
Scope:	`class apext::com::secoc::ComSecOcFvmErrorDomain`
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const noexcept(false) override;`

▽

△

Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	implementation defined	
Description:	Creates a new instance of `apext::com::secoc::ComSecOcFvmException` from `errorCode` and throws it. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

C.2.4 Class: ComSecOcFvmException

[SWS_CM_12512] Definition of API class apext::com::secoc::ComSecOcFvmException

Upstream requirements: RS_AP_00122, RS_AP_00127, RS_AP_00130

⌈

Kind:	class
Header file:	#include "apext/com/secoc/fvm_error_domain.h"
Forwarding header file:	#include "apext/com/com_fwd.h"
Scope:	`namespace apext::com::secoc`
Symbol:	ComSecOcFvmException
Base class:	ara::core::Exception
Syntax:	`class ComSecOcFvmException : public ara::core::Exception {...};`
Description:	Defines a class for exceptions to be thrown by the SecOc Freshness Value Manager

⌋

C.2.4.1 Public Member Functions

C.2.4.1.1 Constructors

C.2.4.1.1.1 ComSecOcFvmException

[SWS_CM_12513] Definition of API function apext::com::secoc::ComSecOcFvmException::ComSecOcFvmException

Upstream requirements: RS_AP_00120, RS_AP_00121, RS_AP_00130, RS_AP_00132

⌈

Kind:	function	
Header file:	#include "apext/com/secoc/fvm_error_domain.h"	
Scope:	`class apext::com::secoc::ComSecOcFvmException`	
Syntax:	`explicit ComSecOcFvmException (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code.
Exception Safety:	exception safe	
Thread Safety:	implementation defined	
Description:	Constructs a new `apext::com::secoc::ComSecOcFvmException` object containing an error code	

⌋

D Not implemented requirements

This functional cluster implements all functional requirements specified in the corresponding requirement specifications.

E History of Constraints and Specification Items

This chapter provides an overview of the history of constraints and specification items. Please note that the lists in this chapter also include constraints and specification items that have been removed from the specification in a later version. These constraints and specification items do not appear as hyperlinks in the document.

E.1 Constraint and Specification Item Changes between AUTOSAR Release R23-11 and R24-11

E.1.1 Added Specification Items in R24-11

Number	Heading
[SWS_CM_00048]	Aborting method calls in case of detected unconfigured application error
[SWS_CM_00049]	Aborting method calls in case of detected unspecified E2E error
[SWS_CM_00050]	Implement reboot detection
[SWS_CM_00051]	Sending SOME/IP SubscribeEventgroup messages - renewal due to detected reboot
[SWS_CM_00052]	NULL value of the variant type field
[SWS_CM_00053]	Definition of API function ara::com::InstanceIdentifier::InstanceIdentifier&&
[SWS_CM_00054]	Definition of API function ara::com::InstanceIdentifier::operator&=
[SWS_CM_00055]	Definition of API function ara::com::InstanceIdentifier::~InstanceIdentifier
[SWS_CM_00056]	Definition of API function ara::com::InstanceIdentifier::InstanceIdentifier&
[SWS_CM_00057]	Mapping of Offer Service in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00058]	Mapping of Stop Offer Service in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00059]	Mapping of Find Service in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00060]	Mapping of Subscribe Service in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00061]	Mapping of Unsubscribe Service in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00062]	Data accumulation on a `RequiredSomeipServiceInstance` in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00063]	Configuration of data accumulation on an `IEEE1722 ACF stream`
[SWS_CM_00064]	Configuration of data accumulation on an `IEEE1722 ACF stream` in case of `collectionTrigger` equals `always`
[SWS_CM_00065]	Configuration of data accumulation on an `IEEE1722 ACF stream` in case of `collectionTrigger` equals `never`
[SWS_CM_00066]	Configuration of no data accumulation on an `IEEE1722 ACF stream`

△

Number	Heading
[SWS_CM_00067]	Configuration of `IEEE1722TpAcfConnection`. `mixedBusTypeCollection`
[SWS_CM_00068]	No method support for `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00069]	Only field notifier support for `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00070]	Transmission of Non-Time-Synchronous Control Format messages
[SWS_CM_00071]	Transmission of Time-Synchronous Control Format messages
[SWS_CM_00072]	Reception of Non-Time-Synchronous Control Format messages
[SWS_CM_00073]	Reception of Time-Synchronous Control Format messages
[SWS_CM_00074]	Ignoring not mapped elements in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00075]	Deserializing incomplete data belonging to a field in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00076]	Unconditional sending of an `IEEE1722 ACF stream` event message
[SWS_CM_00077]	IEEE1722 ACF Protocol used for sending of an `IEEE1722 ACF stream` event message
[SWS_CM_00078]	Destination of an `IEEE1722 ACF stream` event message
[SWS_CM_00079]	Checks for a received signal-based serialized `IEEE1722 ACF stream` event message
[SWS_CM_00080]	Deserializing the `IEEE1722 ACF stream` signal-based serialized payload
[SWS_CM_00081]	Deserializing incomplete data on the proxy side belonging to an event and `eventReceptionDefaultValue` is defined in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00082]	Deserializing incomplete data on the proxy side belonging to an event and `eventReceptionDefaultValue` is not defined in case of `Signal-Based IEEE1722 ACF` network binding
[SWS_CM_00083]	Call StopOfferService() before destruction of Skeleton
[SWS_CM_00084]	Readying the Future of a Method implementation after Skeleton destruction
[SWS_CM_00085]	Precondition for Event/Field unsubscription
[SWS_CM_00086]	Precondition for Skeleton destruction
[SWS_CM_00087]	Precondition for Proxy destruction
[SWS_CM_00088]	Service Identifier String
[SWS_CM_00089]	Serialization of calls to transport fault condition handlers for Events, Triggers, Field Notifiers, Method requests and Field Get and Set requests
[SWS_CM_00090]	Client-side reporting of transport fault conditions for Methods and Field Getters and Setters
[SWS_CM_00091]	Definition of API enum ara::com::e2e::TransportFaultCondition
[SWS_CM_00092]	Definition of API type ara::com::e2e::TransportFaultConditionHandler
[SWS_CM_00093]	Definition of API function `{ <hierarchical-namespace-list-lower-skeleton>}`::skeleton::`{ <service-interface-name-upper-camel>}`Skeleton::SetTransport FaultConditionHandler

▽

△

Number	Heading
[SWS_CM_00094]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::SetTransportFaultConditionHandler
[SWS_CM_00095]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::UnsetTransportFaultConditionHandler
[SWS_CM_00096]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::SetTransportFaultConditionHandler
[SWS_CM_00097]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::SetTransportFaultConditionHandler
[SWS_CM_00098]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::UnsetTransportFaultConditionHandler
[SWS_CM_00099]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetTransportFaultConditionHandler
[SWS_CM_00100]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetTransportFaultConditionHandler
[SWS_CM_00105]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::UnsetTransportFaultConditionHandler
[SWS_CM_00106]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::SetTransportFaultConditionHandler
[SWS_CM_00107]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::SetTransportFaultConditionHandler
[SWS_CM_00108]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::UnsetTransportFaultConditionHandler
[SWS_CM_00109]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetTransportFaultConditionHandler
[SWS_CM_00110]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetTransportFaultConditionHandler
[SWS_CM_00126]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::UnsetTransportFaultConditionHandler

▽

△

Number	Heading
[SWS_CM_00154]	MACsec secure channel between communication nodes
[SWS_CM_00155]	Communication over a MACsec security association
[SWS_CM_00210]	Mapping of Event::GetE2EStateMachineState() for Proxy Events
[SWS_CM_00211]	Mapping of ara::com::e2e::TransportFaultCondition for Proxy Events
[SWS_CM_00212]	Mapping of ara::com::e2e::TransportFaultCondition for Proxy Triggers
[SWS_CM_00213]	Skeleton E2E Error Handler - Invocation
[SWS_CM_00214]	Skeleton E2E Error Handler - Invocation Arguments
[SWS_CM_00215]	Mapping of Method::GetE2EStateMachineState() for Proxy Methods
[SWS_CM_00216]	Mapping of ara::com::e2e::TransportFaultCondition for Proxy Methods
[SWS_CM_00217]	Mapping of Field::GetE2EStateMachineState() for Proxy Field Notifiers
[SWS_CM_00218]	Mapping of ara::com::e2e::TransportFaultCondition for Proxy Field Notifiers
[SWS_CM_00219]	Skeleton E2E Error Handler - Invocation and Invocation Parameters
[SWS_CM_00220]	Mapping of Field::GetE2EStateMachineState() for Proxy Field Getter and Setter
[SWS_CM_00221]	Mapping of ara::com::e2e::TransportFaultCondition for Proxy Field Getter and Setter
[SWS_CM_00726]	Definition of API class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::triggers::{<trigger-name-upper-camel>}
[SWS_CM_00727]	Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}
[SWS_CM_10048]	Definition of Event VerificationStatus.VerificationStatus
[SWS_CM_10049]	Definition of Method VerificationStatusConfigurationByFreshnessId.VerifyStatusOverride
[SWS_CM_10050]	Definition of Method VerificationStatusConfigurationByDataId.VerifyStatusOverride
[SWS_CM_10417]	Register handlers failure due to null pointer or empty function.
[SWS_CM_11377]	Definition of Namespace ara::com::runtime
[SWS_CM_11378]	Definition of API enum apext::com::secoc::FVM::VerificationStatus
[SWS_CM_11379]	ara::com Runtime Header File: file name
[SWS_CM_11500]	Service Common Header File: service namespace
[SWS_CM_11501]	Service Common Header File: common namespace
[SWS_CM_11502]	Service Skeleton Header File: triggers namespace
[SWS_CM_11503]	Service Proxy Header File: file name, includes and multiple inclusion guard
[SWS_CM_11504]	Service Proxy Header File: service namespace
[SWS_CM_11505]	Service Proxy Header File: triggers namespace
[SWS_CM_11506]	Definition of API variable {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}::serviceIdentifier

▽

△

Number	Heading
[SWS_CM_11507]	Definition of API variable {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}::serviceVersion
[SWS_CM_11508]	Definition of API variable {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}::serviceContractVersionMajor
[SWS_CM_11509]	Definition of API variable {<hierarchical-namespace-list-lower-common>}::common::{<si-shortname>}::serviceContractVersionMinor
[SWS_CM_11510]	Definition of API class ara::com::ServiceIdentifierType
[SWS_CM_11511]	Definition of API function ara::com::ServiceIdentifierType::operator==
[SWS_CM_11512]	Definition of API function ara::com::ServiceIdentifierType::operator<
[SWS_CM_11513]	Definition of API function ara::com::ServiceIdentifierType::operator=
[SWS_CM_11514]	Definition of API function ara::com::ServiceIdentifierType::toString
[SWS_CM_11515]	Definition of API class ara::com::ServiceVersionType
[SWS_CM_11516]	Definition of API function ara::com::ServiceVersionType::operator==
[SWS_CM_11517]	Definition of API function ara::com::ServiceVersionType::operator<
[SWS_CM_11518]	Definition of API function ara::com::ServiceVersionType::operator=
[SWS_CM_11519]	Definition of API function ara::com::ServiceVersionType::ToString
[SWS_CM_11520]	Definition of API function ara::com::InstanceIdentifier::Create
[SWS_CM_11521]	Definition of API function ara::com::InstanceIdentifier::InstanceIdentifier
[SWS_CM_11522]	Definition of API function ara::com::InstanceIdentifier::toString
[SWS_CM_11523]	Definition of API function ara::com::InstanceIdentifier::operator==
[SWS_CM_11524]	Definition of API function ara::com::InstanceIdentifier::operator<
[SWS_CM_11525]	Definition of API function ara::com::InstanceIdentifier::operator=
[SWS_CM_11526]	Definition of API function ara::com::FindServiceHandle::operator==
[SWS_CM_11527]	Definition of API function ara::com::FindServiceHandle::operator<
[SWS_CM_11528]	Definition of API function ara::com::FindServiceHandle::operator=
[SWS_CM_11529]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::operator==
[SWS_CM_11530]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::operator<
[SWS_CM_11531]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::GetInstanceId

▽

△

Number	Heading
[SWS_CM_11532]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::operator=
[SWS_CM_11533]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::operator=
[SWS_CM_11534]	Definition of API function ara::com::SamplePtr::SamplePtr
[SWS_CM_11535]	Definition of API function ara::com::SamplePtr::SamplePtr
[SWS_CM_11536]	Definition of API function ara::com::SamplePtr::SamplePtr
[SWS_CM_11537]	Definition of API function ara::com::SamplePtr::SamplePtr
[SWS_CM_11538]	Definition of API function ara::com::SamplePtr::operator=
[SWS_CM_11539]	Definition of API function ara::com::SamplePtr::operator=
[SWS_CM_11540]	Definition of API function ara::com::SamplePtr::operator=
[SWS_CM_11541]	Definition of API function ara::com::SamplePtr::operator*
[SWS_CM_11542]	Definition of API function ara::com::SamplePtr::operator->
[SWS_CM_11543]	Definition of API function ara::com::SamplePtr::operator bool
[SWS_CM_11544]	Definition of API function ara::com::SamplePtr::Swap
[SWS_CM_11545]	Definition of API function ara::com::SamplePtr::Reset
[SWS_CM_11546]	Definition of API function ara::com::SamplePtr::Get
[SWS_CM_11547]	Definition of API function ara::com::SamplePtr::~SamplePtr
[SWS_CM_11548]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::operator=
[SWS_CM_11549]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::operator=
[SWS_CM_11550]	Definition of API variable {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<method-out-arg-symbol>}
[SWS_CM_11551]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::operator=
[SWS_CM_11552]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::operator=
[SWS_CM_11554]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::GetE2EStateMachineState

▽

△

Number	Heading
[SWS_CM_11601]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::Subscribe
[SWS_CM_11602]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::Unsubscribe
[SWS_CM_11603]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::GetFreeSampleCount
[SWS_CM_11604]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::GetSubscriptionState
[SWS_CM_11605]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetReceiveHandler
[SWS_CM_11606]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::UnsetReceiveHandler
[SWS_CM_11607]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetSubscriptionStateChangeHandler
[SWS_CM_11608]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetSubscriptionStateChangeHandler
[SWS_CM_11609]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::UnsetSubscriptionStateChangeHandler
[SWS_CM_11610]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::GetNewSamples
[SWS_CM_11611]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::SetReceiveHandler
[SWS_CM_11612]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::GetE2EStateMachineState
[SWS_CM_11614]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::UnsetReceiveHandler
[SWS_CM_11615]	Definition of API type ara::com::FieldReceiveHandler
[SWS_CM_12023]	Timeout time in message segmentation for Events
[SWS_CM_12024]	Timeout time in message segmentation for Method Call
[SWS_CM_12025]	Timeout time in message segmentation for Method Response
[SWS_CM_12026]	Mapping of ara::com::e2e::ComE2EErrc
[SWS_CM_19999]	Definition of API function apext::com::secoc::FVM::SetVerificationStatus

▽

△

Number	Heading
[SWS_CM_80514]	Deserializing incomplete data on the proxy side belonging to a statically defined event and `eventReceptionDefaultValue` is defined
[SWS_CM_80515]	Deserializing incomplete data on the proxy side belonging to a statically defined event and `eventReceptionDefaultValue` is not defined
[SWS_CM_98444]	`Service Proxy Header File`: fields namespace
[SWS_CM_98447]	`Service Proxy Header File`: events namespace
[SWS_CM_99332]	Definition of API class `{<hierarchical-namespace-list-lower-proxy>}`::proxy::methods::`{<fnfmethod-name-upper-camel>}`
[SWS_CM_99333]	Definition of API class `{<hierarchical-namespace-list-lower-proxy>}`::proxy::methods::`{<method-name-upper-camel>}`::Output
[SWS_CM_99444]	Definition of API function `{<hierarchical-namespace-list-lower-proxy>}`::proxy::`{<service-interface-name-upper-camel>}`Proxy::~`{<service-interface-name-upper-camel>}`Proxy
[SWS_CM_99445]	Definition of API variable `{<hierarchical-namespace-list-lower-proxy>}`::proxy::`{<service-interface-name-upper-camel>}`Proxy::`{<event-name-upper-camel>}`
[SWS_CM_99446]	Definition of API variable `{<hierarchical-namespace-list-lower-proxy>}`::proxy::`{<service-interface-name-upper-camel>}`Proxy::`{<field-name-upper-camel>}`
[SWS_CM_99447]	Definition of API variable `{<hierarchical-namespace-list-lower-proxy>}`::proxy::`{<service-interface-name-upper-camel>}`Proxy::`{<method-name-upper-camel>}`
[SWS_CM_99556]	Definition of API class `{<hierarchical-namespace-list-lower-skeleton>}`::skeleton::`{<service-interface-name-upper-camel>}`Skeleton::`{<method-name-upper-camel>}`Output
[SWS_CM_99557]	Definition of API variable `{<hierarchical-namespace-list-lower-skeleton>}`::skeleton::`{<service-interface-name-upper-camel>}`Skeleton::`{<event-name-upper-camel>}`
[SWS_CM_99558]	Definition of API variable `{<hierarchical-namespace-list-lower-skeleton>}`::skeleton::`{<service-interface-name-upper-camel>}`Skeleton::`{<field-name-upper-camel>}`
[SWS_CM_99559]	Definition of API variable `{<hierarchical-namespace-list-lower-skeleton>}`::skeleton::`{<service-interface-name-upper-camel>}`Skeleton::`{<trigger-name-upper-camel>}`

Table E.1: Added Specification Items in R24-11

E.1.2 Changed Specification Items in R24-11

Number	Heading
[SWS_CM_00002]	Definition of API class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton
[SWS_CM_00003]	Service Skeleton event class
[SWS_CM_00004]	Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy
[SWS_CM_00005]	Service Proxy event class
[SWS_CM_00006]	Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<method-name-upper-camel>}
[SWS_CM_00007]	Definition of API class {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}
[SWS_CM_00008]	Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}
[SWS_CM_00038]	E2E_check for method request provides Result with SMState and ara::com::e2e::ComE2EErrc
[SWS_CM_00047]	E2E Error Handler - Invocation Arguments
[SWS_CM_00101]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::OfferService
[SWS_CM_00111]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::StopOfferService
[SWS_CM_00112]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::Get
[SWS_CM_00113]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::Set
[SWS_CM_00114]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::RegisterGetHandler
[SWS_CM_00116]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::RegisterSetHandler
[SWS_CM_00118]	Definition of API function ara::com::runtime::ResolveInstanceIDs

△

Number	Heading
[SWS_CM_00119]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::Update
[SWS_CM_00122]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::FindService
[SWS_CM_00123]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StartFindService
[SWS_CM_00125]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StopFindService
[SWS_CM_00130]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton
[SWS_CM_00131]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::{<service-interface-name-upper-camel>}Proxy
[SWS_CM_00134]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton
[SWS_CM_00135]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton
[SWS_CM_00136]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::{<service-interface-name-upper-camel>}Proxy
[SWS_CM_00137]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::{<service-interface-name-upper-camel>}Proxy
[SWS_CM_00141]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::Subscribe
[SWS_CM_00151]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::Unsubscribe
[SWS_CM_00152]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton

▽

△

Number	Heading
[SWS_CM_00153]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<service-interface-name-upper-camel>}Skeleton
[SWS_CM_00162]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::Send
[SWS_CM_00181]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetReceiveHandler
[SWS_CM_00183]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::UnsetReceiveHandler
[SWS_CM_00191]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<method-name-upper-camel>}
[SWS_CM_00196]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<method-name-upper-camel>}::operator()
[SWS_CM_00199]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::ProcessNextMethodCall
[SWS_CM_00209]	Start of service discovery protocol on Client side
[SWS_CM_00226]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::GetNewTriggers
[SWS_CM_00250]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::SetReceiveHandler
[SWS_CM_00301]	Definition of API enum ara::com::MethodCallProcessingMode
[SWS_CM_00302]	Definition of API class ara::com::InstanceIdentifier
[SWS_CM_00303]	Definition of API class ara::com::FindServiceHandle
[SWS_CM_00304]	Definition of API type ara::com::ServiceHandleContainer
[SWS_CM_00306]	Definition of API class ara::com::SamplePtr
[SWS_CM_00308]	Definition of API type ara::com::SampleAllocateePtr
[SWS_CM_00309]	Definition of API type ara::com::EventReceiveHandler
[SWS_CM_00310]	Definition of API enum ara::com::SubscriptionState
[SWS_CM_00311]	Definition of API type ara::com::SubscriptionStateChangeHandler
[SWS_CM_00312]	Definition of API class {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType

▽

△

Number	Heading
[SWS_CM_00316]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::GetSubscriptionState
[SWS_CM_00317]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::HandleType
[SWS_CM_00318]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::HandleType
[SWS_CM_00319]	Definition of API type ara::com::InstanceIdentifierContainer
[SWS_CM_00333]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetSubscriptionStateChangeHandler
[SWS_CM_00334]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::UnsetSubscriptionStateChangeHandler
[SWS_CM_00349]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::HandleType
[SWS_CM_00351]	Definition of API type ara::com::TriggerReceiveHandler
[SWS_CM_00352]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::SetReceiveHandler
[SWS_CM_00353]	Definition of API function ara::com::FindServiceHandle::FindServiceHandle
[SWS_CM_00383]	Definition of API type ara::com::FindServiceHandler
[SWS_CM_00622]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::FindService
[SWS_CM_00623]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StartFindService
[SWS_CM_00701]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::GetNewSamples
[SWS_CM_00705]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::GetFreeSampleCount
[SWS_CM_00721]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::triggers::{<trigger-name-upper-camel>}::Send

▽

△

Number	Heading
[SWS_CM_00723]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::Subscribe
[SWS_CM_00810]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::triggers::{<trigger-name-upper-camel>}::Unsubscribe
[SWS_CM_01002]	Service Skeleton Header File: file name, includes and multiple inclusion guard
[SWS_CM_01005]	Service Skeleton Header File: service namespace
[SWS_CM_01006]	Service Skeleton Header File: skeleton namespace
[SWS_CM_01007]	Service Proxy Header File: proxy namespace
[SWS_CM_01009]	Service Skeleton Header File: events namespace
[SWS_CM_01010]	Service Common Header File Service Identifier and Service Contract Version
[SWS_CM_01012]	Service Common Header File: file name, includes and multiple inclusion guard
[SWS_CM_01013]	ara::com Types Header File: file name
[SWS_CM_01015]	Service Proxy Header File: methods namespace
[SWS_CM_01018]	ara::com Types Header File: namespace
[SWS_CM_01031]	Service Skeleton Header File: fields namespace
[SWS_CM_01071]	Definition of API enum ara::com::ServiceState
[SWS_CM_01072]	Definition of API type ara::com::ServiceStateHandler
[SWS_CM_01073]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::GetServiceState
[SWS_CM_01074]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::SetServiceState ChangeHandler
[SWS_CM_01075]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::UnsetServiceState ChangeHandler
[SWS_CM_01076]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::SetServiceState ChangeHandler
[SWS_CM_09004]	Adding Service IDs, Service Instance IDs, and ServiceInterface Contract Versions to the DDS DomainParticipant's USER_DATA QoS Policy
[SWS_CM_10312]	Content of the SOME/IP response message
[SWS_CM_10357]	Distinguishing errors from normal responses
[SWS_CM_10358]	Identifying the right application error in a message with Message Type set to RESPONSE (0x80)

▽

△

Number	Heading
[SWS_CM_10383]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::GetHandle
[SWS_CM_10429]	Identifying the right application error in a message with Message Type set to ERROR (0x81)
[SWS_CM_10430]	Handling invalid messages with Message Type set to ERROR (0x81)
[SWS_CM_10431]	Mapping of ara::core::ErrorCode
[SWS_CM_10432]	Definition of API enum ara::com::ComErrc
[SWS_CM_10438]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::Create
[SWS_CM_10467]	Wrong Method Call Processing Mode Error for ServiceSkeleton constructor
[SWS_CM_10470]	E2E Error Handler - Existence
[SWS_CM_10471]	E2E Error Handler - Invocation Arguments
[SWS_CM_10474]	Definition of API enum ara::com::e2e::ComE2EErrc
[SWS_CM_10475]	GetE2EStateMachineState method for Events
[SWS_CM_10493]	Local Access Control Activation
[SWS_CM_10494]	Remote Access Control Activation
[SWS_CM_10497]	Authentication Failure
[SWS_CM_10524]	Mapping Triggers to DDS Topics
[SWS_CM_10525]	DDS Topic data type definition
[SWS_CM_10526]	Mapping of Send method
[SWS_CM_10527]	Mapping of Subscribe method
[SWS_CM_10528]	Creating a DDS DataReader for trigger subscription
[SWS_CM_10529]	Defining a DDS DataReaderListener
[SWS_CM_10530]	Mapping of Unsubscribe method
[SWS_CM_10531]	Mapping of GetSubscriptionState() method
[SWS_CM_10532]	Mapping of GetNewTriggers method
[SWS_CM_10534]	Mapping of SetReceiveHandler method
[SWS_CM_10535]	Mapping of UnsetReceiveHandler() method
[SWS_CM_10536]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_10537]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_10539]	Local access control on receiving triggers
[SWS_CM_11001]	Mapping of OfferService method
[SWS_CM_11005]	Mapping of StopOfferService method
[SWS_CM_11006]	Mapping of FindService method
[SWS_CM_11007]	Finding a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_11008]	Creating a DDS DomainParticipant suitable for performing client-side operations

▽

△

Number	Heading
[SWS_CM_11009]	Discovering remote Service Instances through DDS DomainParticipants
[SWS_CM_11011]	Defining a DDS BuiltinParticipantListener
[SWS_CM_11012]	Binding a BuiltinParticipantListener to a DDS DomainParticipant
[SWS_CM_11013]	Mapping of `StopFindService()` method
[SWS_CM_11014]	Unbinding a BuiltinParticipantListener from a DDS DomainParticipant
[SWS_CM_11015]	Mapping Events to DDS Topics
[SWS_CM_11016]	DDS Topic data type definition
[SWS_CM_11017]	Mapping of Send method
[SWS_CM_11018]	Mapping of Subscribe method
[SWS_CM_11019]	Creating a DDS DataReader for event subscription
[SWS_CM_11020]	Defining a DDS DataReaderListener
[SWS_CM_11021]	Mapping of Unsubscribe method
[SWS_CM_11022]	Mapping of `GetSubscriptionState()` method
[SWS_CM_11023]	Mapping of GetNewSamples method
[SWS_CM_11024]	Mapping of GetFreeSampleCount method
[SWS_CM_11025]	Mapping of SetReceiveHandler method
[SWS_CM_11026]	Mapping of `UnsetReceiveHandler()` method
[SWS_CM_11027]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_11028]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_11041]	DDS serialization of `StdCppImplementationDataType` of `category` `VALUE`
[SWS_CM_11042]	DDS serialization of enumeration data types
[SWS_CM_11043]	DDS serialization of `StdCppImplementationDataType` of `category` `STRUCTURE`
[SWS_CM_11044]	DDS serialization of `StdCppImplementationDataType` of `category` `STRING` with string `shortName`
[SWS_CM_11046]	Encoding Format and Endianness of Strings in DDS
[SWS_CM_11047]	DDS serialization of `CppImplementationDataType` of `category` `VECTOR`
[SWS_CM_11048]	DDS serialization of `CppImplementationDataType` of `category` `ARRAY`
[SWS_CM_11049]	DDS serialization of `CppImplementationDataType` of `category` `ASSOCIATIVE_MAP`
[SWS_CM_11050]	DDS serialization of `CppImplementationDataType` of `category` `VARIANT`
[SWS_CM_11100]	Mapping Methods to DDS Service Methods and Topics
[SWS_CM_11101]	DDS Service Request Topic data type definition
[SWS_CM_11102]	DDS Service Reply Topic data type definition
[SWS_CM_11103]	Creating a DataWriter to handle method requests on the client side
[SWS_CM_11104]	Creating a DataReader to handle method responses on the client side
[SWS_CM_11105]	Creating a DataReader to handle method requests on the server side

▽

△

Number	Heading
[SWS_CM_11106]	Creating a DataWriter to handle method responses on the server side
[SWS_CM_11107]	Calling a service method from the client side
[SWS_CM_11108]	Notifying the client of a response to a method call
[SWS_CM_11109]	Processing a method call on the server side (event driven)
[SWS_CM_11110]	Creating a DataReaderListener to process asynchronous requests on the server side
[SWS_CM_11111]	Processing a method call on the server side (polling)
[SWS_CM_11112]	Sending a method call response from the server side
[SWS_CM_11130]	Mapping Fields with hasNotifier attribute to DDS Topics
[SWS_CM_11131]	Field Notifier DDS Topic data type definition
[SWS_CM_11132]	Mapping of Update method
[SWS_CM_11133]	Mapping of Subscribe method
[SWS_CM_11134]	Creating a DDS DataReader for field subscription
[SWS_CM_11135]	Creating a DDS DataReaderListener for field subscription
[SWS_CM_11136]	Mapping of Unsubscribe method
[SWS_CM_11144]	Mapping of Field Get/Set methods to DDS Service Methods and Topics
[SWS_CM_11145]	DDS Service Request Topic data type definition for Field getter and setter operations
[SWS_CM_11146]	DDS Service Reply Topic data type definition for Field getter and setter operations
[SWS_CM_11147]	Creating a DataWriter to handle get/set requests on the client side
[SWS_CM_11148]	Creating a DataReader to handle get/set responses on the client side
[SWS_CM_11149]	Creating a DataReader to handle get/set requests on the server side
[SWS_CM_11150]	Creating a DataWriter to handle get/set responses on the server side
[SWS_CM_11151]	Calling get/set method associated with a field from the client side
[SWS_CM_11152]	Notifying the client of the response to the get/set method call
[SWS_CM_11153]	Processing a get/set method call associated with a field on the server side (event driven)
[SWS_CM_11154]	Creating a DataReaderListener to process asynchronous requests for field getters and setters on the server side
[SWS_CM_11155]	Processing a get/set method call associated with a field on the server side (polling)
[SWS_CM_11156]	Sending a response for a get/set method call associated with a field from the server side
[SWS_CM_11273]	Initialization of the FVM
[SWS_CM_11274]	SecOC secure channel sending
[SWS_CM_11275]	SecOC secure message build attempts
[SWS_CM_11276]	SecOC secure channel reception
[SWS_CM_11277]	SecOC secure message verification attempts
[SWS_CM_11280]	Definition of ServiceInterface VerificationStatus

▽

△

Number	Heading
[SWS_CM_11281]	Definition of ServiceInterface VerificationStatusConfigurationByFreshnessId
[SWS_CM_11282]	Definition of ServiceInterface VerificationStatusConfigurationByDataId
[SWS_CM_11286]	Definition of API class apext::com::secoc::FVContainer
[SWS_CM_11287]	Definition of API class apext::com::secoc::FVM
[SWS_CM_11288]	Definition of API function apext::com::secoc::FVM::GetRxFreshness
[SWS_CM_11289]	Definition of API function apext::com::secoc::FVM::GetTxFreshness
[SWS_CM_11290]	Definition of API function apext::com::secoc::FVM::Initialize
[SWS_CM_11327]	Definition of API class ara::com::ComException
[SWS_CM_11328]	Definition of API function ara::com::ComException::ComException
[SWS_CM_11329]	Definition of API class ara::com::ComErrorDomain
[SWS_CM_11330]	Definition of API function ara::com::ComErrorDomain::ComErrorDomain
[SWS_CM_11331]	Definition of API function ara::com::ComErrorDomain::Name
[SWS_CM_11332]	Definition of API function ara::com::ComErrorDomain::Message
[SWS_CM_11333]	Definition of API function ara::com::ComErrorDomain::ThrowAsException
[SWS_CM_11334]	Definition of API function ara::com::GetComErrorDomain
[SWS_CM_11335]	Definition of API function ara::com::MakeErrorCode
[SWS_CM_11336]	Definition of API type ara::com::ComErrorDomain::Errc
[SWS_CM_11337]	Definition of API type ara::com::ComErrorDomain::Exception
[SWS_CM_11342]	Definition of API enum apext::com::secoc::ComSecOcFvmErrc
[SWS_CM_11344]	Definition of API variable apext::com::secoc::FVContainer::length
[SWS_CM_11345]	Definition of API variable apext::com::secoc::FVContainer::value
[SWS_CM_11352]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StartFindService
[SWS_CM_11354]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetSubscriptionStateChangeHandler
[SWS_CM_11356]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SetReceiveHandler
[SWS_CM_11360]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::RegisterGetHandler
[SWS_CM_11362]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::RegisterSetHandler
[SWS_CM_11365]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::StartFindService

▽

△

Number	Heading
[SWS_CM_11370]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::~{<service-interface-name-upper-camel>}Skeleton
[SWS_CM_11371]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::{<service-interface-name-upper-camel>}Proxy::HandleType::~HandleType
[SWS_CM_11400]	Definition of API type {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::SampleType
[SWS_CM_11401]	Definition of API type {<hierarchical-namespace-list-lower-proxy>}::proxy::events::{<event-name-upper-camel>}::SampleType
[SWS_CM_11402]	Definition of API type {<hierarchical-namespace-list-lower-skeleton>}::skeleton::fields::{<field-name-upper-camel>}::FieldType
[SWS_CM_11403]	Definition of API type {<hierarchical-namespace-list-lower-proxy>}::proxy::fields::{<field-name-upper-camel>}::FieldType
[SWS_CM_12008]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::SetSubscriptionStateChangeHandler
[SWS_CM_12009]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::SetSubscriptionStateChangeHandler
[SWS_CM_12010]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::UnsetSubscriptionStateChangeHandler
[SWS_CM_12011]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::GetSubscriptionState
[SWS_CM_12501]	Definition of API class ara::com::e2e::ComE2EException
[SWS_CM_12502]	Definition of API function ara::com::e2e::ComE2EException::ComE2EException
[SWS_CM_12503]	Definition of API class ara::com::e2e::ComE2EErrorDomain
[SWS_CM_12504]	Definition of API type ara::com::e2e::ComE2EErrorDomain::Errc
[SWS_CM_12505]	Definition of API type ara::com::e2e::ComE2EErrorDomain::Exception
[SWS_CM_12506]	Definition of API function ara::com::e2e::ComE2EErrorDomain::ComE2EErrorDomain
[SWS_CM_12507]	Definition of API function ara::com::e2e::ComE2EErrorDomain::Name

▽

△

Number	Heading
[SWS_CM_12508]	Definition of API function ara::com::e2e::ComE2EErrorDomain::Message
[SWS_CM_12509]	Definition of API function ara::com::e2e::ComE2EErrorDomain::ThrowAs Exception
[SWS_CM_12510]	Definition of API function ara::com::e2e::GetComE2EErrorDomain
[SWS_CM_12511]	Definition of API function ara::com::e2e::MakeErrorCode
[SWS_CM_12512]	Definition of API class apext::com::secoc::ComSecOcFvmException
[SWS_CM_12513]	Definition of API function apext::com::secoc::ComSecOcFvm Exception::ComSecOcFvmException
[SWS_CM_12514]	Definition of API class apext::com::secoc::ComSecOcFvmErrorDomain
[SWS_CM_12515]	Definition of API type apext::com::secoc::ComSecOcFvmErrorDomain::Errc
[SWS_CM_12516]	Definition of API type apext::com::secoc::ComSecOcFvmError Domain::Exception
[SWS_CM_12517]	Definition of API function apext::com::secoc::ComSecOcFvmError Domain::ComSecOcFvmErrorDomain
[SWS_CM_12518]	Definition of API function apext::com::secoc::ComSecOcFvmError Domain::Name
[SWS_CM_12519]	Definition of API function apext::com::secoc::ComSecOcFvmError Domain::Message
[SWS_CM_12520]	Definition of API function apext::com::secoc::ComSecOcFvmError Domain::ThrowAsException
[SWS_CM_12521]	Definition of API function apext::com::secoc::GetComSecOcFvmError Domain
[SWS_CM_12522]	Definition of API function apext::com::secoc::MakeErrorCode
[SWS_CM_90001]	Local access control on executing methods
[SWS_CM_90003]	Local access control on receiving events
[SWS_CM_90006]	Local access control on service discovery
[SWS_CM_90411]	E2E_check for Events provides Result with `ara::com::e2e::SMState` and `ara::com::e2e::ProfileCheckStatus`
[SWS_CM_90416]	E2E_check Result on a null sample
[SWS_CM_90420]	Definition of API function ara::com::SamplePtr::GetProfileCheckStatus
[SWS_CM_90421]	Definition of API enum ara::com::e2e::ProfileCheckStatus
[SWS_CM_90422]	Definition of API enum ara::com::e2e::SMState
[SWS_CM_90434]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::{<service-interface-name-upper-camel>}Skeleton::{<fnfmethod-name-upper-camel>}
[SWS_CM_90435]	Definition of API function {<hierarchical-namespace-list-lower-proxy>}::proxy::methods::{<fnfmethod-name-upper-camel>}::operator()
[SWS_CM_90437]	Definition of API function {<hierarchical-namespace-list-lower-skeleton>}::skeleton::events::{<event-name-upper-camel>}::Send

▽

△

Number	Heading
[SWS_CM_90438]	Definition of API function `{ <hierarchical-namespace-list-lower-skeleton> }`::skeleton::events::`{<event-name-upper-camel>}`::Allocate
[SWS_CM_90477]	E2E Error Return Code
[SWS_CM_90478]	E2E_check for method response provides Result with `ara::com::e2e::SMState` and `ara::com::e2e::ComE2EErrc`
[SWS_CM_90503]	Assigning a DDS DomainParticipant to a Service Instance
[SWS_CM_90508]	Advertising Service IDs, Service Instance IDs, and ServiceInterface Contract Versions over the `ara.com://services/discovery` topic
[SWS_CM_90509]	Mapping of StopOfferService method
[SWS_CM_90510]	Mapping of FindService method
[SWS_CM_90511]	Finding a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_90512]	Creating a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_90513]	Discovering remote Service Instances through the ara.com://services/ discovery topic
[SWS_CM_90514]	Mapping of StartFindService method
[SWS_CM_90515]	Mapping of `StopFindService()` method
[SWS_CM_99040]	MACsec secure channel between communication nodes and MACsec security association

Table E.2: Changed Specification Items in R24-11

E.1.3 Deleted Specification Items in R24-11

Number	Heading
[SWS_CM_00009]	Re-entrancy and thread-safety - General
[SWS_CM_00010]	Re-entrancy and thread-safety - `OfferService`
[SWS_CM_00011]	Re-entrancy and thread-safety- `StopOfferService`
[SWS_CM_00012]	Re-entrancy and thread-safety - `Send`
[SWS_CM_00013]	Re-entrancy and thread-safety - `Allocate`
[SWS_CM_00014]	Re-entrancy and thread-safety - `RegisterGetHandler`
[SWS_CM_00015]	Re-entrancy and thread-safety - `RegisterSetHandler`
[SWS_CM_00016]	Re-entrancy and thread-safety - `Update`
[SWS_CM_00017]	Re-entrancy and thread-safety - `ServiceSkeleton` method implementation
[SWS_CM_00018]	Re-entrancy and thread-safety - `FindService`
[SWS_CM_00019]	Re-entrancy and thread-safety - `StartFindService`

▽

△

Number	Heading
[SWS_CM_00020]	Re-entrancy and thread-safety - `StopFindService`
[SWS_CM_00021]	Re-entrancy and thread-safety - `GetHandle`
[SWS_CM_00022]	Re-entrancy and thread-safety - `Subscribe`
[SWS_CM_00023]	Re-entrancy and thread-safety - `Unsubscribe`
[SWS_CM_00024]	Re-entrancy and thread-safety - `GetSubscriptionState`
[SWS_CM_00025]	Re-entrancy and thread-safety - `SetSubscriptionStateChangeHandler`
[SWS_CM_00026]	Re-entrancy and thread-safety - `UnsetSubscriptionStateChangeHandler`
[SWS_CM_00027]	Re-entrancy and thread-safety - `GetFreeSampleCount`
[SWS_CM_00028]	Re-entrancy and thread-safety - `SetReceiveHandler`
[SWS_CM_00029]	Re-entrancy and thread-safety - `UnsetReceiveHandler`
[SWS_CM_00030]	Re-entrancy and thread-safety - `Get`
[SWS_CM_00031]	Re-entrancy and thread-safety - `Set`
[SWS_CM_00032]	Re-entrancy and thread-safety - `Method` call operator
[SWS_CM_00035]	Re-entrancy and thread-safety - `Unsubscribe`
[SWS_CM_00115]	Existence of RegisterGetHandler method
[SWS_CM_00117]	Existence of the RegisterSetHandler method
[SWS_CM_00132]	Existence of getter method
[SWS_CM_00133]	Existence of the set method
[SWS_CM_00198]	Set service method processing mode
[SWS_CM_00249]	Service `Trigger` reception trigger
[SWS_CM_00251]	Error in `Trigger SetReceiveHandler`
[SWS_CM_00702]	Signature of Callable f
[SWS_CM_00704]	Return Value
[SWS_CM_00706]	Return Value of GetFreeSampleCount
[SWS_CM_00714]	Re-entrancy and thread-safety - `GetNewSamples`
[SWS_CM_00722]	Re-entrancy and thread-safety - `Send`
[SWS_CM_00724]	Re-entrancy and thread-safety - `Subscribe`
[SWS_CM_00725]	Errors in Send trigger call
[SWS_CM_00820]	Binding information
[SWS_CM_00821]	Service location scenarios
[SWS_CM_01001]	Inclusion of Types header file
[SWS_CM_01004]	Inclusion of common header file
[SWS_CM_01008]	Namespace for Service Identifier Type definitions
[SWS_CM_01017]	Service Identifier Type definitions in Common header file
[SWS_CM_01019]	Data Type declarations in Types header file
[SWS_CM_01020]	Common/Service header files directory structure
[SWS_CM_01050]	`Variant` Class Template

▽

△

Number	Heading
[SWS_CM_01051]	`Variant` default constructor
[SWS_CM_01052]	`Variant` move constructor
[SWS_CM_01053]	`Variant` copy constructor
[SWS_CM_01054]	`Variant` converting constructor
[SWS_CM_01055]	`Variant` explicit converting constructor with specified alternative
[SWS_CM_01056]	`Variant` explicit converting constructor with specified alternative and initializer list
[SWS_CM_01057]	`Variant` explicit converting constructor with alternative specified by index
[SWS_CM_01058]	`Variant` explicit converting constructor with alternative specified by index and initializer list
[SWS_CM_01059]	`Variant` destructor
[SWS_CM_01060]	`Variant` move assignment operator
[SWS_CM_01061]	`Variant` default copy assignment operator
[SWS_CM_01062]	`Variant` converting assignment operator
[SWS_CM_01063]	`Variant` function to return the zero-based index of the alternative
[SWS_CM_01064]	`Variant` function to check if the Variant is in invalid state
[SWS_CM_01065]	`Variant` function to swap two Variants
[SWS_CM_01066]	`Variant` function to create a new value in-place, in an existing Variant object
[SWS_CM_01067]	`Variant` function to create a new value in-place, in an existing Variant object using an initializer list
[SWS_CM_01068]	`Variant` function to create a new value in-place, in an existing Variant object by destoying and initializing the contained value
[SWS_CM_01069]	`Variant` function to create a new value in-place, in an existing Variant object by destoying and initializing the contained value using an initializer list
[SWS_CM_10226]	Serialized Variant size
[SWS_CM_10250]	Data type for the length field of variants
[SWS_CM_10362]	Raising checked errors for application errors
[SWS_CM_10370]	Common header file for Application Errors
[SWS_CM_10372]	Inclusion of Implementation Types header files
[SWS_CM_10435]	Exception-less creation of service skeleton using Instance ID
[SWS_CM_10436]	Exception-less creation of service skeleton using Instance Spec
[SWS_CM_10437]	Exception-less creation of service skeleton using Instance ID Container
[SWS_CM_10492]	IAM Module Instantiation
[SWS_CM_11251]	Re-entrancy and thread-safety - `GetNewTriggers`
[SWS_CM_11264]	Definition general ara::com errors
[SWS_CM_11266]	Definition of Application Errors
[SWS_CM_11267]	General errors domain
[SWS_CM_11326]	Creation of an object using Named Constructor approach
[SWS_CM_11340]	Definition general ara::com::secoc errors

▽

△

Number	Heading
[SWS_CM_11341]	SecOcFvm errors domain
[SWS_CM_11350]	Execution Context for process service method invocation
[SWS_CM_11351]	Error behaviour of provided Execution Context for process service method invocation
[SWS_CM_11353]	Error behavior of provided Execution Context for finding service with handler registration using Instance ID
[SWS_CM_11355]	Error behaviour of provided Execution Context for setting Subscription State change handler
[SWS_CM_11357]	Error behaviour of provided Execution Context for enabling service event trigger
[SWS_CM_11361]	Error behaviour of provided Execution Context for registering Getters
[SWS_CM_11363]	Error behaviour of provided Execution Context for registering Setters
[SWS_CM_11364]	Minimal behaviour of provided Execution Context
[SWS_CM_11366]	Error behavior of provided Execution Context for finding service with handler registration using InstanceSpecifier
[SWS_CM_12000]	Implementation types header files directory structure
[SWS_CM_12001]	C++ Implementation Data Types files
[SWS_CM_12016]	Re-entrancy and thread-safety GetSubscriptionState
[SWS_CM_12017]	Re-entrancy and thread-safety SetSubscriptionStateChangeHandler
[SWS_CM_12018]	Re-entrancy and thread-safety UnsetSubscriptionStateChangeHandler
[SWS_CM_12020]	StdCppImplementationDataTypes with category VALUE supported for serialization
[SWS_CM_90426]	Mapping of ProfileCheckStatus
[SWS_CM_90427]	Mapping of SMState
[SWS_CM_90431]	GetE2EStateMachineState shall provide the global SMState
[SWS_CM_99000]	CommunicationGroupServer Service
[SWS_CM_99001]	Broadcast method of CommunicationGroupServer Service
[SWS_CM_99002]	Peer To Peer Message method of CommunicationGroupServer Service
[SWS_CM_99007]	CommunicationGroupClient Service
[SWS_CM_99008]	Message method of CommunicationGroupClient Service
[SWS_CM_99009]	Message Response event of CommunicationGroupClient Service
[SWS_CM_99010]	Broadcast task
[SWS_CM_99011]	Peer To Peer message task
[SWS_CM_99012]	Message Response task
[SWS_CM_99013]	List Clients task
[SWS_CM_99014]	Message Response event of CommunicationGroupServer Service
[SWS_CM_99015]	List Clients method of CommunicationGroupServer Service
[SWS_CM_99016]	Connection Status of a Communication Group Server
[SWS_CM_99017]	category value COMMUNICATION_GROUP
[SWS_CM_99018]	category value COMMUNICATION_GROUP_SERVER

▽

△

Number	Heading
[SWS_CM_99019]	`category` value `COMMUNICATION_GROUP_CLIENT`
[SWS_CM_99020]	Communcation Group `template`
[SWS_CM_99021]	`SHORT-NAME` value of generated `CommunicationGroupServer` service
[SWS_CM_99022]	`SHORT-NAME` value of generated `CommunicationGroupClient` service
[SWS_CM_99023]	Definition general ara::com::cg errors
[SWS_CM_99024]	Definition of API enum ara::com::cg::CgErrc
[SWS_CM_99026]	E2E errors domain
[SWS_CM_99027]	Cg errors domain
[SWS_CM_99028]	Types of APIs - Communication and Service Discovery APIs
[SWS_CM_99034]	
[SWS_CM_99041]	Lifetime of data samples pointed to by `SamplePtr`
[SWS_CM_99042]	Dereferencing a dangling `SamplePtr`
[SWS_CM_99043]	Destroying, resetting, assigning to, and swapping a dangling `SamplePtr`
[SWS_CM_99044]	Lifetime of data samples pointed to by `SampleAllocateePtr`
[SWS_CM_99045]	Dereferencing a dangling `SampleAllocateePtr`
[SWS_CM_99046]	Destroying, resetting, releasing, assigning to, and swapping a dangling `SampleAllocateePtr`
[SWS_CM_99047]	Lifetime of data samples pointed to by `SamplePtr` - Unsubscribe
[SWS_CM_99048]	Lifetime of data samples pointed to by `SamplePtr` - StopOfferService

Table E.3: Deleted Specification Items in R24-11

E.1.4 Added Constraints in R24-11

Number	Heading
[SWS_CM_-CONSTR_-00008]	Configurable Namespace
[SWS_CM_-CONSTR_-00009]	No service discovery for Signal-Based IEEE1722 ACF Network binding

Table E.4: Added Constraints in R24-11

E.1.5 Changed Constraints in R24-11

none

E.1.6 Deleted Constraints in R24-11

Number	Heading
[SWS_CM_-CONSTR_-00006]	SOME/IP Service Discovery Reboot detection

Table E.5: Deleted Constraints in R24-11

E.2 Constraint and Specification Item Changes between AUTOSAR Release R22-11 and R23-11

E.2.1 Added Specification Items in R23-11

Number	Heading
[SWS_CM_00033]	Payload of the E2E Error Response
[SWS_CM_00034]	E2E Error Handler - Invocation
[SWS_CM_00036]	Deserialization of the data according to the network binding for method request
[SWS_CM_00037]	E2E Protection header removal from serialized data for method requests
[SWS_CM_00038]	E2E_check for method request provides Result with SMState and ProfileCheckStatus
[SWS_CM_00039]	Argument dataID in E2E_check for method requests
[SWS_CM_00040]	Processing the non-E2E-protected header of E2E-protected method request
[SWS_CM_00041]	E2E-protected Methods Arguments Serialization
[SWS_CM_00042]	GetProfileCheckStatus method of SamplePtr
[SWS_CM_00043]	Argument dataID in E2E_check for events without serialized sample
[SWS_CM_00044]	E2E Protection header removal from serialized data
[SWS_CM_00045]	Argument dataID in E2E_check for event with serialized sample
[SWS_CM_00046]	E2E protection of events in Send
[SWS_CM_00047]	E2E Error Handler - Invocation Arguments
[SWS_CM_00250]	`Trigger SetReceiveHandler`
[SWS_CM_00251]	Error in `Trigger SetReceiveHandler`
[SWS_CM_00270]	Maximum number of vector elements
[SWS_CM_00349]	HandleType default constructor shall be deleted
[SWS_CM_00352]	Execution Context for enabling service Trigger trigger
[SWS_CM_00353]	The default constructor of FindServiceHandle shall be deleted
[SWS_CM_00725]	Errors in Send trigger call
[SWS_CM_01076]	
[SWS_CM_10542]	Local access control on providing service instances

▽

△

Number	Heading
[SWS_CM_10543]	Remote access control on providing service instances
[SWS_CM_11365]	Execution Context for finding service with handler registration using Instance Specifier
[SWS_CM_11366]	Error behavior of provided Execution Context for finding service with handler registration using InstanceSpecifier
[SWS_CM_11367]	Definition of Port VerificationStatus provided by functional cluster CM
[SWS_CM_11368]	Definition of Port VerificationStatusConfigurationByFreshnessId provided by functional cluster CM
[SWS_CM_11369]	Definition of Port VerificationStatusConfigurationByDataId provided by functional cluster CM
[SWS_CM_12002]	Active subscriber
[SWS_CM_12003]	Active subscriber when SOME/IP Network binding is used
[SWS_CM_12004]	Deserializing incomplete data on the proxy side belonging to an event and `eventReceptionDefaultValue` is defined
[SWS_CM_12005]	Deserializing incomplete data on the proxy side belonging to an event and `eventReceptionDefaultValue` is not defined
[SWS_CM_12006]	Asynchronous nature of Subscribe()
[SWS_CM_12007]	New data samples received by CM at execution time of receive handler
[SWS_CM_12008]	Set Subscription State change handler on Skeleton side
[SWS_CM_12009]	Execution Context for setting Subscription State change handler on Skeleton side
[SWS_CM_12010]	Unset Subscription State change handler on Skeleton side
[SWS_CM_12011]	Query Subscription State on Skeleton side
[SWS_CM_12012]	Subscription State change handler
[SWS_CM_12013]	Call SubscriptionStateChangeHandler on Skeleton side with kSubscribed
[SWS_CM_12014]	Call SubscriptionStateChangeHandler on Skeleton side with kNotSubscribed
[SWS_CM_12015]	Query Subscription State on Skeleton side
[SWS_CM_12016]	Re-entrancy and thread-safety GetSubscriptionState
[SWS_CM_12017]	Re-entrancy and thread-safety SetSubscriptionStateChangeHandler
[SWS_CM_12018]	Re-entrancy and thread-safety UnsetSubscriptionStateChangeHandler
[SWS_CM_12019]	Service Discovery Endpoint Options
[SWS_CM_12020]	`StdCppImplementationDataType`s with category VALUE supported for serialization
[SWS_CM_12021]	Mapping of `ProfileCheckStatus`
[SWS_CM_12022]	Mapping of `SMState`
[SWS_CM_80104]	Deserializing more data than expected
[SWS_CM_99041]	Lifetime of data samples pointed to by `SamplePtr`
[SWS_CM_99042]	Dereferencing a dangling `SamplePtr`
[SWS_CM_99043]	Destroying, resetting, assigning to, and swapping a dangling `SamplePtr`
[SWS_CM_99044]	Lifetime of data samples pointed to by `SampleAllocateePtr`
[SWS_CM_99045]	Dereferencing a dangling `SampleAllocateePtr`

▽

△

Number	Heading
[SWS_CM_99046]	Destroying, resetting, releasing, assigning to, and swapping a dangling `SampleAllocateePtr`
[SWS_CM_99047]	Lifetime of data samples pointed to by `SamplePtr` - Unsubscribe
[SWS_CM_99048]	Lifetime of data samples pointed to by `SamplePtr` - StopOfferService

Table E.6: Added Specification Items in R23-11

E.2.2 Changed Specification Items in R23-11

Number	Heading
[SWS_CM_00101]	Method to offer a service
[SWS_CM_00112]	Method to get the value of a field
[SWS_CM_00113]	Method to set the value of a field
[SWS_CM_00114]	Registering Getters
[SWS_CM_00116]	Registering Setters
[SWS_CM_00118]	Definition of API function ara::com::runtime::ResolveInstanceIDs
[SWS_CM_00119]	Update Function
[SWS_CM_00123]	Find service with handler registration using Instance ID
[SWS_CM_00129]	Ensuring the existence of SetHandler
[SWS_CM_00141]	Method to subscribe to a service event
[SWS_CM_00151]	Method to unsubscribe from a service event
[SWS_CM_00162]	Send event where application is responsible for the data
[SWS_CM_00181]	Enable service event trigger
[SWS_CM_00183]	Disable service event trigger
[SWS_CM_00196]	Initiate a method call
[SWS_CM_00199]	Process Service method invocation
[SWS_CM_00209]	Start of service discovery protocol on Client side
[SWS_CM_00226]	Method to update the trigger counter
[SWS_CM_00249]	Service `Trigger` reception trigger
[SWS_CM_00301]	Method Call Processing Mode
[SWS_CM_00302]	Instance Identifier Class
[SWS_CM_00303]	Find Service Handle
[SWS_CM_00304]	Service Handle Container
[SWS_CM_00306]	Sample Pointer
[SWS_CM_00308]	Sample Allocatee Pointer
[SWS_CM_00309]	Event Receive Handler
[SWS_CM_00310]	Subscription State
[SWS_CM_00311]	Subscription State Changed Handler

▽

△

Number	Heading
[SWS_CM_00313]	Call SubscriptionStateChangeHandler with kSubscriptionPending on the Proxy side
[SWS_CM_00314]	Call SubscriptionStateChangeHandler with kSubscribed on the Proxy side
[SWS_CM_00316]	Query Subscription State on Proxy side
[SWS_CM_00319]	Instance Identifier Container Class
[SWS_CM_00333]	Set Subscription State change handler on the Proxy side
[SWS_CM_00334]	Unset Subscription State change handler on Proxy side
[SWS_CM_00351]	Trigger Receive Handler
[SWS_CM_00383]	Find Service Handler
[SWS_CM_00623]	Find service with handler registration using Instance Specifier
[SWS_CM_00710]	No implicit context switches
[SWS_CM_00711]	GetNewSamples shall provide data samples if GetFreeSampleCount is not 0
[SWS_CM_00721]	Send trigger
[SWS_CM_00723]	Method to subscribe to a service trigger
[SWS_CM_01019]	Data Type declarations in Types header file
[SWS_CM_01073]	
[SWS_CM_01074]	
[SWS_CM_01075]	
[SWS_CM_09004]	Adding Service IDs, Service Instance IDs, and ServiceInterface Contract Versions to the DDS DomainParticipant's USER_DATA QoS Policy
[SWS_CM_10036]	Serialization of supported primitive `StdCppImplementationDataType`s
[SWS_CM_10088]	Default Serialization layout of Variants specified by the union data type in SOME/IP
[SWS_CM_10172]	Payload Byte order definition
[SWS_CM_10240]	Session handling state
[SWS_CM_10287]	Conditions for sending of a SOME/IP event message
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10296]	Invoke receive handler
[SWS_CM_10319]	Conditions for sending of a SOME/IP event message
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10381]	Sending SOME/IP SubscribeEventgroup messages - renewal
[SWS_CM_10431]	Mapping of ara::core::ErrorCode
[SWS_CM_10432]	Definition of API enum ara::com::ComErrc
[SWS_CM_10435]	Exception-less creation of service skeleton using Instance ID
[SWS_CM_10436]	Exception-less creation of service skeleton using Instance Spec
[SWS_CM_10437]	Exception-less creation of service skeleton using Instance ID Container
[SWS_CM_10438]	Exception-less creation of service proxy
[SWS_CM_10460]	Options of E2E Protection for Methods
[SWS_CM_10462]	E2E-protected Methods Request Message Protection

▽

△

Number	Heading
[SWS_CM_10463]	E2E-protected Method Requests dataID Argument
[SWS_CM_10464]	E2E protection header according to the network binding in the method request
[SWS_CM_10465]	E2E counter of method response shall match with the one in method request
[SWS_CM_10466]	E2E checking of the method request in ServiceSkeleton (message reception)
[SWS_CM_10467]	Wrong Method Call Processing Mode Error for ServiceSkeleton named constructor
[SWS_CM_10468]	E2E checking of the method request in ServiceSkeleton (ProcessNextMethodCall)
[SWS_CM_10469]	Argument dataId in E2E_protect for methods
[SWS_CM_10471]	E2E Error Handler - Invocation Arguments
[SWS_CM_10472]	E2E Error Response
[SWS_CM_10474]	Definition of API enum ara::com::e2e::E2EErrc
[SWS_CM_10475]	GetE2EStateMachineState method for Events
[SWS_CM_10511]	Conditions for sending of a SOME/IP trigger
[SWS_CM_10512]	Content of the SOME/IP trigger
[SWS_CM_10515]	Silently discarding SOME/IP triggers for unsubscribed triggers
[SWS_CM_10516]	Invoke receive handler
[SWS_CM_10517]	Failures in sending a SOME/IP trigger
[SWS_CM_10518]	Conditions for sending of a trigger
[SWS_CM_10519]	Content of the SOME/IP serialized trigger message
[SWS_CM_10524]	Mapping Triggers to DDS Topics
[SWS_CM_10525]	DDS Topic data type definition
[SWS_CM_10526]	Mapping of Send method
[SWS_CM_10527]	Mapping of Subscribe method
[SWS_CM_10528]	Creating a DDS DataReader for trigger subscription
[SWS_CM_10529]	Defining a DDS DataReaderListener
[SWS_CM_10530]	Mapping of Unsubscribe method
[SWS_CM_10531]	Mapping of GetSubscriptionState method
[SWS_CM_10532]	Mapping of GetNewTriggers method
[SWS_CM_10534]	Mapping of SetReceiveHandler method
[SWS_CM_10535]	Mapping of UnsetReceiveHandler method
[SWS_CM_10536]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_10537]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_10539]	Local access control on receiving triggers
[SWS_CM_10550]	Assigning a DDS Topic and a DDS DataWriter to every Trigger in the ServiceInterface
[SWS_CM_11000]	DDS Compliance
[SWS_CM_11001]	Mapping of OfferService method

▽

△

Number	Heading
[SWS_CM_11002]	Assigning a DDS DomainParticipant to a Service Instance
[SWS_CM_11003]	Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface
[SWS_CM_11005]	Mapping of StopOfferService method
[SWS_CM_11006]	Mapping of FindService method
[SWS_CM_11007]	Finding a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_11008]	Creating a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_11009]	Discovering remote Service Instances through DDS DomainParticipants
[SWS_CM_11010]	Mapping of StartFindService method
[SWS_CM_11011]	Defining a DDS BuiltinParticipantListener
[SWS_CM_11012]	Binding a BuiltinParticipantListener to a DDS DomainParticipant
[SWS_CM_11013]	Mapping of StopFindService method
[SWS_CM_11014]	Unbinding a BuiltinParticipantListener from a DDS DomainParticipant
[SWS_CM_11015]	Mapping Events to DDS Topics
[SWS_CM_11016]	DDS Topic data type definition
[SWS_CM_11017]	Mapping of Send method
[SWS_CM_11018]	Mapping of Subscribe method
[SWS_CM_11019]	Creating a DDS DataReader for event subscription
[SWS_CM_11020]	Defining a DDS DataReaderListener
[SWS_CM_11021]	Mapping of Unsubscribe method
[SWS_CM_11022]	Mapping of GetSubscriptionState method
[SWS_CM_11023]	Mapping of GetNewSamples method
[SWS_CM_11024]	Mapping of GetFreeSampleCount method
[SWS_CM_11025]	Mapping of SetReceiveHandler method
[SWS_CM_11026]	Mapping of UnsetReceiveHandler method
[SWS_CM_11027]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_11028]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_11029]	Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Methods in the ServiceInterface
[SWS_CM_11030]	Assigning a DDS Topic and a DDS DataWriter to every Field in the ServiceInterface with its hasNotifier attribute equal to true
[SWS_CM_11031]	Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Field Getters/Setters in the ServiceInterface
[SWS_CM_11040]	DDS standard serialization rules
[SWS_CM_11041]	DDS serialization of `StdCppImplementationDataType` of `category` `VALUE`
[SWS_CM_11042]	DDS serialization of enumeration data types
[SWS_CM_11043]	DDS serialization of `StdCppImplementationDataType` of `category` `STRUCTURE`

▽

△

Number	Heading
[SWS_CM_11044]	DDS serialization of `StdCppImplementationDataType` of `category` `STRING` with string `shortName`
[SWS_CM_11046]	Encoding Format and Endianness of Strings in DDS
[SWS_CM_11047]	DDS serialization of `CppImplementationDataType` of `category` `VECTOR`
[SWS_CM_11048]	DDS serialization of `CppImplementationDataType` of `category` `ARRAY`
[SWS_CM_11049]	DDS serialization of `CppImplementationDataType` of `category` `ASSOCIATIVE_MAP`
[SWS_CM_11050]	DDS serialization of `CppImplementationDataType` of `category` `VARIANT`
[SWS_CM_11100]	Mapping Methods to DDS Service Methods and Topics
[SWS_CM_11101]	DDS Service Request Topic data type definition
[SWS_CM_11102]	DDS Service Reply Topic data type definition
[SWS_CM_11103]	Creating a DataWriter to handle method requests on the client side
[SWS_CM_11104]	Creating a DataReader to handle method responses on the client side
[SWS_CM_11105]	Creating a DataReader to handle method requests on the server side
[SWS_CM_11106]	Creating a DataWriter to handle method responses on the server side
[SWS_CM_11107]	Calling a service method from the client side
[SWS_CM_11108]	Notifying the client of a response to a method call
[SWS_CM_11109]	Processing a method call on the server side (event driven)
[SWS_CM_11110]	Creating a DataReaderListener to process asynchronous requests on the server side
[SWS_CM_11111]	Processing a method call on the server side (polling)
[SWS_CM_11112]	Sending a method call response from the server side
[SWS_CM_11130]	Mapping Fields with hasNotifier attribute to DDS Topics
[SWS_CM_11131]	Field Notifier DDS Topic data type definition
[SWS_CM_11132]	Mapping of Update method
[SWS_CM_11133]	Mapping of Subscribe method
[SWS_CM_11134]	Creating a DDS DataReader for field subscription
[SWS_CM_11135]	Creating a DDS DataReaderListener for field subscription
[SWS_CM_11136]	Mapping of Unsubscribe method
[SWS_CM_11137]	Mapping of GetSubscriptionState method
[SWS_CM_11138]	Mapping of GetNewSamples method
[SWS_CM_11139]	Mapping of GetFreeSampleCount method
[SWS_CM_11140]	Mapping of SetReceiveHandler method
[SWS_CM_11141]	Mapping of UnsetReceiveHandler method
[SWS_CM_11142]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_11143]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_11144]	Mapping of Field Get/Set methods to DDS Service Methods and Topics

▽

△

Number	Heading
[SWS_CM_11145]	DDS Service Request Topic data type definition for Field getter and setter operations
[SWS_CM_11146]	DDS Service Reply Topic data type definition for Field getter and setter operations
[SWS_CM_11147]	Creating a DataWriter to handle get/set requests on the client side
[SWS_CM_11148]	Creating a DataReader to handle get/set responses on the client side
[SWS_CM_11149]	Creating a DataReader to handle get/set requests on the server side
[SWS_CM_11150]	Creating a DataWriter to handle get/set responses on the server side
[SWS_CM_11151]	Calling get/set method associated with a field from the client side
[SWS_CM_11152]	Notifying the client of the response to the get/set method call
[SWS_CM_11153]	Processing a get/set method call associated with a field on the server side (event driven)
[SWS_CM_11154]	Creating a DataReaderListener to process asynchronous requests for field getters and setters on the server side
[SWS_CM_11155]	Processing a get/set method call associated with a field on the server side (polling)
[SWS_CM_11156]	Sending a response for a get/set method call associated with a field from the server side
[SWS_CM_11273]	Initialization of the FVM
[SWS_CM_11274]	SecOC secure channel sending
[SWS_CM_11276]	SecOC secure channel reception
[SWS_CM_11326]	Creation of an object using Named Constructor approach
[SWS_CM_11330]	Definition of API function ara::com::ComErrorDomain::ComErrorDomain
[SWS_CM_11331]	Definition of API function ara::com::ComErrorDomain::Name
[SWS_CM_11332]	Definition of API function ara::com::ComErrorDomain::Message
[SWS_CM_11333]	Definition of API function ara::com::ComErrorDomain::ThrowAsException
[SWS_CM_11334]	Definition of API function ara::com::GetComErrorDomain
[SWS_CM_11335]	Definition of API function ara::com::MakeErrorCode
[SWS_CM_11336]	Definition of API type ara::com::ComErrorDomain::Errc
[SWS_CM_11337]	Definition of API type ara::com::ComErrorDomain::Exception
[SWS_CM_11340]	Definition general ara::com::secoc errors
[SWS_CM_11341]	SecOcFvm errors domain
[SWS_CM_11342]	Definition of API enum ara::com::secoc::ComSecOcFvmErrc
[SWS_CM_11344]	Definition of API variable ara::com::secoc::FVContainer::length
[SWS_CM_11345]	Definition of API variable ara::com::secoc::FVContainer::value
[SWS_CM_11352]	Execution Context for finding service with handler registration using Instance ID
[SWS_CM_11360]	Execution Context for registering Getters
[SWS_CM_11362]	Execution Context for registering Setters
[SWS_CM_11412]	Deserializing incomplete data on the proxy side

▽

△

Number	Heading
[SWS_CM_12000]	Implementation types header files directory structure
[SWS_CM_12001]	C++ Implementation Data Types files
[SWS_CM_12501]	Definition of API class ara::com::e2e::E2EException
[SWS_CM_12502]	Definition of API function ara::com::e2e::E2EException::E2EException
[SWS_CM_12503]	Definition of API class ara::com::e2e::E2EErrorDomain
[SWS_CM_12504]	Definition of API type ara::com::e2e::E2EErrorDomain::Errc
[SWS_CM_12505]	Definition of API type ara::com::e2e::E2EErrorDomain::Exception
[SWS_CM_12506]	Definition of API function ara::com::e2e::E2EErrorDomain::E2EErrorDomain
[SWS_CM_12507]	Definition of API function ara::com::e2e::E2EErrorDomain::Name
[SWS_CM_12508]	Definition of API function ara::com::e2e::E2EErrorDomain::Message
[SWS_CM_12509]	Definition of API function ara::com::e2e::E2EErrorDomain::ThrowAs Exception
[SWS_CM_12510]	Definition of API function ara::com::e2e::GetE2EErrorDomain
[SWS_CM_12511]	Definition of API function ara::com::e2e::MakeErrorCode
[SWS_CM_12512]	Definition of API class ara::com::secoc::ComSecOcFvmException
[SWS_CM_12513]	Definition of API function ara::com::secoc::ComSecOcFvmException::Com SecOcFvmException
[SWS_CM_12514]	Definition of API class ara::com::secoc::ComSecOcFvmErrorDomain
[SWS_CM_12515]	Definition of API type ara::com::secoc::ComSecOcFvmErrorDomain::Errc
[SWS_CM_12516]	Definition of API type ara::com::secoc::ComSecOcFvmError Domain::Exception
[SWS_CM_12517]	Definition of API function ara::com::secoc::ComSecOcFvmError Domain::ComSecOcFvmErrorDomain
[SWS_CM_12518]	Definition of API function ara::com::secoc::ComSecOcFvmError Domain::Name
[SWS_CM_12519]	Definition of API function ara::com::secoc::ComSecOcFvmError Domain::Message
[SWS_CM_12520]	Definition of API function ara::com::secoc::ComSecOcFvmError Domain::ThrowAsException
[SWS_CM_12521]	Definition of API function ara::com::secoc::GetComSecOcFvmErrorDomain
[SWS_CM_12522]	Definition of API function ara::com::secoc::MakeErrorCode
[SWS_CM_80021]	Conditions for sending of an event message
[SWS_CM_80025]	Content of the SOME/IP serialized event message
[SWS_CM_80063]	Conditions for sending of an event message
[SWS_CM_80067]	Content of the SOME/IP serialized event message
[SWS_CM_90001]	Local access control on executing methods
[SWS_CM_90003]	Local access control on receiving events
[SWS_CM_90006]	Local access control on service discovery
[SWS_CM_90108]	SecOC secure channel for methods using reliable transport
[SWS_CM_90109]	SecOC secure channel for events and triggers using reliable transport

▽

△

Number	Heading
[SWS_CM_90110]	SecOC secure channel for fields
[SWS_CM_90115]	SecOC secure channel for methods using unreliable transport
[SWS_CM_90116]	SecOC secure channel for events and triggers using unreliable transport
[SWS_CM_90435]	Initiate a `Fire and Forget method` call
[SWS_CM_90437]	Send event where Communication Management is responsible for the data
[SWS_CM_90489]	Argument sourceID in E2E_check for method requests
[SWS_CM_90490]	Argument messageType in E2E_check for method requests
[SWS_CM_90491]	Argument messageResult E2E_check for method requests
[SWS_CM_90492]	Argument sourceId in E2E_protect for methods
[SWS_CM_90493]	Argument messageType in E2E_protect for methods
[SWS_CM_90494]	Argument messageResult STD_MESSAGERESULT_OK in E2E_protect for methods
[SWS_CM_99000]	`CommunicationGroupServer` Service
[SWS_CM_99001]	Broadcast method of `CommunicationGroupServer` Service
[SWS_CM_99002]	Peer To Peer Message method of `CommunicationGroupServer` Service
[SWS_CM_99007]	`CommunicationGroupClient` Service
[SWS_CM_99008]	Message method of `CommunicationGroupClient` Service
[SWS_CM_99009]	Message Response event of `CommunicationGroupClient` Service
[SWS_CM_99010]	Broadcast task
[SWS_CM_99011]	Peer To Peer message task
[SWS_CM_99012]	Message Response task
[SWS_CM_99013]	List Clients task
[SWS_CM_99014]	Message Response event of `CommunicationGroupServer` Service
[SWS_CM_99015]	List Clients method of `CommunicationGroupServer` Service
[SWS_CM_99016]	Connection Status of a Communication Group Server
[SWS_CM_99017]	`category` value `COMMUNICATION_GROUP`
[SWS_CM_99018]	`category` value `COMMUNICATION_GROUP_SERVER`
[SWS_CM_99019]	`category` value `COMMUNICATION_GROUP_CLIENT`
[SWS_CM_99020]	Communcation Group `template`
[SWS_CM_99021]	`SHORT-NAME` value of generated `CommunicationGroupServer` service
[SWS_CM_99022]	`SHORT-NAME` value of generated `CommunicationGroupClient` service
[SWS_CM_99023]	Definition general ara::com::cg errors
[SWS_CM_99024]	Definition of API enum ara::com::cg::CgErrc
[SWS_CM_99027]	Cg errors domain
[SWS_CM_99035]	Subscription State change handler on the Proxy side

Table E.7: Changed Specification Items in R23-11

E.2.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_CM_00228]	Return Value
[SWS_CM_10013]	Header Byte order
[SWS_CM_10360]	Failures in sending a SOME/IP event message
[SWS_CM_10391]	Serializing Scale Linear And Texttable Data Type
[SWS_CM_10476]	Defining a RawDataStream
[SWS_CM_10477]	Connect stream link
[SWS_CM_10478]	Shutdown stream link
[SWS_CM_10479]	Read data from stream
[SWS_CM_10480]	Write data to stream
[SWS_CM_10481]	
[SWS_CM_10482]	
[SWS_CM_10483]	
[SWS_CM_10484]	
[SWS_CM_10485]	
[SWS_CM_10486]	
[SWS_CM_10487]	
[SWS_CM_10488]	Raw data stream header file existence
[SWS_CM_10489]	Raw data stream header file namespace
[SWS_CM_10490]	Data Type declarations in Raw data stream header file
[SWS_CM_10499]	Remote access control on providing methods
[SWS_CM_10500]	Remote access control on providing events
[SWS_CM_10502]	Remote access control on providing field notifiers
[SWS_CM_10503]	Remote access control on providing field setters
[SWS_CM_10504]	Remote access control on providing field getters
[SWS_CM_10538]	Restrictions on sending triggers
[SWS_CM_10540]	Remote access control on providing triggers
[SWS_CM_11268]	Definition general ara::com::raw errors
[SWS_CM_11291]	
[SWS_CM_11292]	
[SWS_CM_11293]	
[SWS_CM_11295]	
[SWS_CM_11296]	
[SWS_CM_11297]	
[SWS_CM_11298]	
[SWS_CM_11299]	
[SWS_CM_11300]	

▽

△

Number	Heading
[SWS_CM_11301]	
[SWS_CM_11302]	
[SWS_CM_11303]	
[SWS_CM_11304]	
[SWS_CM_11305]	
[SWS_CM_11306]	
[SWS_CM_11307]	
[SWS_CM_11309]	
[SWS_CM_11310]	
[SWS_CM_11311]	
[SWS_CM_11312]	
[SWS_CM_11313]	
[SWS_CM_11314]	
[SWS_CM_11315]	
[SWS_CM_11316]	
[SWS_CM_11317]	
[SWS_CM_11318]	
[SWS_CM_11319]	
[SWS_CM_11320]	
[SWS_CM_11322]	
[SWS_CM_11323]	
[SWS_CM_11324]	
[SWS_CM_11325]	
[SWS_CM_11358]	Execution Context to update the event cache
[SWS_CM_11359]	Error behaviour of provided Execution Context to update the event cache
[SWS_CM_12367]	
[SWS_CM_90002]	Restrictions on sending events
[SWS_CM_90005]	Restrictions on offering services
[SWS_CM_90007]	Restrictions on using RawDataStreams
[SWS_CM_90211]	Secure UDP and TCP channel creation for TLS and DTLS
[SWS_CM_90212]	Using secure TLS, DTLS channels
[SWS_CM_90213]	TLS secure channel for raw data streams using reliable transport
[SWS_CM_90214]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90215]	IPsec secure channel between communication nodes and Transport of Raw Data Stream communication over an IPsec security association
[SWS_CM_90216]	Socket Options configuration
[SWS_CM_90217]	TLS properties configuration
[SWS_CM_90453]	
[SWS_CM_90454]	

▽

△

Number	Heading
[SWS_CM_90455]	
[SWS_CM_90456]	
[SWS_CM_90457]	
[SWS_CM_90458]	
[SWS_CM_90459]	
[SWS_CM_90460]	
[SWS_CM_90461]	
[SWS_CM_90462]	
[SWS_CM_90463]	
[SWS_CM_90464]	E2E Error Handler - Invocation
[SWS_CM_90465]	E2E Error Handler - Invocation Arguments
[SWS_CM_90466]	Payload of the E2E Error Response
[SWS_CM_99004]	Ethernet endpoint configuration
[SWS_CM_99005]	Wait for incoming connections
[SWS_CM_99006]	Timeout handling
[SWS_CM_99025]	Raw errors domain

Table E.8: Deleted Specification Items in R23-11

E.2.4 Added Constraints in R23-11

Number	Heading
[SWS_CM_-CONSTR_-00002]	SOME/IP Service Discovery Discardable flag
[SWS_CM_-CONSTR_-00003]	SOME/IP Service Discovery Configuration options
[SWS_CM_-CONSTR_-00004]	SOME/IP Service Discovery Load balancing options
[SWS_CM_-CONSTR_-00005]	SOME/IP Service Discovery SD endpoint options
[SWS_CM_-CONSTR_-00006]	SOME/IP Service Discovery Reboot detection

▽

△

Number	Heading
[SWS_CM_-CONSTR_-00007]	E2E Protection

Table E.9: Added Constraints in R23-11

E.2.5 Changed Constraints in R23-11

Number	Heading
[SWS_CM_-CONSTR_-00001]	Optional method arguments with SOME/IP Tag-Length-Value serialization

Table E.10: Changed Constraints in R23-11

E.2.6 Deleted Constraints in R23-11

none

E.3 Constraint and Specification Item Changes between AUTOSAR Release R21-11 and R22-11

E.3.1 Added Specification Items in R22-11

Number	Heading
[SWS_CM_00820]	Binding information
[SWS_CM_00821]	Service location scenarios
[SWS_CM_01071]	
[SWS_CM_01072]	
[SWS_CM_01073]	
[SWS_CM_01074]	
[SWS_CM_01075]	
[SWS_CM_02201]	Static service connection
[SWS_CM_02202]	Service Discovery is bypassed by static service connection
[SWS_CM_02203]	Service versioning is not checked at runtime in case of a static service connection

▽

△

Number	Heading
[SWS_CM_10061]	Supported encoding of CppImplementationDataType with category equal to STRING
[SWS_CM_11372]	SecOC secure channel reception bypass
[SWS_CM_11373]	Cyclic interval of OfferService messages
[SWS_CM_11374]	Periodic link state monitoring
[SWS_CM_11375]	Link loss on Client side
[SWS_CM_11376]	Link loss on Server side
[SWS_CM_11400]	Service skeleton SampleType type alias
[SWS_CM_11401]	Service proxy SampleType type alias
[SWS_CM_11402]	Service skeleton FieldType class
[SWS_CM_11403]	Service proxy FieldType alias
[SWS_CM_11411]	Deserializing incomplete data on the skeleton side
[SWS_CM_11412]	Deserializing incomplete data on the proxy side
[SWS_CM_11413]	Deserializing incomplete data on the proxy side belonging to a field and initValue not defined
[SWS_CM_12501]	
[SWS_CM_12502]	
[SWS_CM_12503]	
[SWS_CM_12504]	
[SWS_CM_12505]	
[SWS_CM_12506]	
[SWS_CM_12507]	
[SWS_CM_12508]	
[SWS_CM_12509]	
[SWS_CM_12510]	
[SWS_CM_12511]	
[SWS_CM_12512]	
[SWS_CM_12513]	
[SWS_CM_12514]	
[SWS_CM_12515]	
[SWS_CM_12516]	
[SWS_CM_12517]	
[SWS_CM_12518]	
[SWS_CM_12519]	
[SWS_CM_12520]	
[SWS_CM_12521]	
[SWS_CM_12522]	
[SWS_CM_99029]	Service Contract Version
[SWS_CM_99030]	Find Service Handle

▽

△

Number	Heading
[SWS_CM_99031]	Send event where application is responsible for the data
[SWS_CM_99032]	Send event where Communication Management is responsible for the data
[SWS_CM_99033]	Allocating data for event transfer
[SWS_CM_99034]	
[SWS_CM_99035]	Set Subscription State change handler
[SWS_CM_99036]	Event message separation time
[SWS_CM_99037]	Method request message separation time
[SWS_CM_99038]	Method response message segmentation
[SWS_CM_99039]	Method response message separation time
[SWS_CM_99040]	MACsec secure channel between communication nodes and MACsec security association

Table E.11: Added Specification Items in R22-11

E.3.2 Changed Specification Items in R22-11

Number	Heading
[SWS_CM_00008]	Service proxy Field class
[SWS_CM_00009]	Re-entrancy and thread-safety - General
[SWS_CM_00103]	Network binding where a service is offered
[SWS_CM_00104]	Network binding for StopOfferService
[SWS_CM_00112]	Method to get the value of a field
[SWS_CM_00113]	Method to set the value of a field
[SWS_CM_00114]	Registering Getters
[SWS_CM_00116]	Registering Setters
[SWS_CM_00118]	
[SWS_CM_00119]	Update Function
[SWS_CM_00122]	Find service with immediately returned request using Instance ID
[SWS_CM_00123]	Find service with handler registration using Instance ID
[SWS_CM_00130]	Creation of service skeleton using Instance ID
[SWS_CM_00152]	Creation of service skeleton using Instance Spec
[SWS_CM_00153]	Creation of service skeleton using Instance ID Container
[SWS_CM_00162]	Send event where application is responsible for the data
[SWS_CM_00198]	Set service method processing mode
[SWS_CM_00199]	Process Service method invocation
[SWS_CM_00201]	Start of service discovery protocol on Server side
[SWS_CM_00202]	SOME/IP FindService message

▽

△

Number	Heading
[SWS_CM_00203]	SOME/IP OfferService message
[SWS_CM_00204]	SOME/IP StopOffer message
[SWS_CM_00205]	Content of SOME/IP SubscribeEventgroup message
[SWS_CM_00206]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_00207]	Content of SOME/IP StopSubscribeEventgroup message
[SWS_CM_00208]	SOME/IP SubscribeEventgroupNack message
[SWS_CM_00209]	Start of service discovery protocol on Client side
[SWS_CM_00227]	Sequence of actions in GetNewTriggers
[SWS_CM_00302]	Instance Identifier Class
[SWS_CM_00303]	Find Service Handle
[SWS_CM_00313]	Call SubscriptionStateChangeHandler with kSubscriptionPending
[SWS_CM_00314]	Call SubscriptionStateChangeHandler with kSubscribed
[SWS_CM_00315]	Re-establishing an active subscription
[SWS_CM_00319]	Instance Identifier Container Class
[SWS_CM_00333]	Set Subscription State change handler
[SWS_CM_00383]	Find Service Handler
[SWS_CM_00622]	Find service with immediately returned request using Instance Specifier
[SWS_CM_00700]	Ensure memory allocation of maxSampleCount samples
[SWS_CM_00703]	Sequence of actions in GetNewSamples
[SWS_CM_00704]	Return Value
[SWS_CM_00707]	Calculation of Free Sample Count
[SWS_CM_01010]	Service Identifier and Service Contract Version
[SWS_CM_10017]	Deserializing incomplete data on the proxy side belonging to a field and initValue defined
[SWS_CM_10036]	Serialization of supported StdCppImplementationDataTypes
[SWS_CM_10202]	Version blocklist
[SWS_CM_10247]	Deserialization of strings
[SWS_CM_10287]	Conditions for sending of a SOME/IP event message
[SWS_CM_10288]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10289]	Source of a SOME/IP event message
[SWS_CM_10290]	Destination of a SOME/IP event message
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10293]	Identifying the right event
[SWS_CM_10297]	Conditions for sending of a SOME/IP request message
[SWS_CM_10300]	Destination of a SOME/IP request message
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10303]	Identifying the right method

▽

△

Number	Heading
[SWS_CM_10312]	Content of the SOME/IP response message
[SWS_CM_10313]	Checks for a received SOME/IP response message
[SWS_CM_10314]	Identifying the right method
[SWS_CM_10317]	Making the Future ready
[SWS_CM_10319]	Conditions for sending of a SOME/IP event message
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10325]	Identifying the right event
[SWS_CM_10328]	Invoke receive handler
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10335]	Identifying the right method
[SWS_CM_10344]	Content of the SOME/IP response message
[SWS_CM_10346]	Identifying the right method
[SWS_CM_10371]	Context of return checked errors
[SWS_CM_10377]	Sending SOME/IP SubscribeEventgroup messages - initial
[SWS_CM_10379]	Silently discarding SOME/IP event messages for unsubscribed events
[SWS_CM_10416]	Reception of a malformed message
[SWS_CM_10432]	
[SWS_CM_10435]	Exception-less creation of service skeleton using Instance ID
[SWS_CM_10436]	Exception-less creation of service skeleton using Instance Spec
[SWS_CM_10437]	Exception-less creation of service skeleton using Instance ID Container
[SWS_CM_10440]	Aborting method calls in case of locally detected failures
[SWS_CM_10454]	Event message segmentation
[SWS_CM_10455]	Method request message segmentation
[SWS_CM_10459]	Legacy string serialization
[SWS_CM_10467]	
[SWS_CM_10474]	
[SWS_CM_10481]	
[SWS_CM_10482]	
[SWS_CM_10483]	
[SWS_CM_10484]	
[SWS_CM_10485]	
[SWS_CM_10486]	
[SWS_CM_10487]	
[SWS_CM_10495]	TLS-based Authentication
[SWS_CM_10496]	IP and IPsec-based Authentication
[SWS_CM_10497]	Authentication Failure
[SWS_CM_10511]	Conditions for sending of a SOME/IP trigger

▽

△

Number	Heading
[SWS_CM_10512]	Content of the SOME/IP trigger
[SWS_CM_10513]	Checks for a received SOME/IP trigger
[SWS_CM_10514]	Identifying the right trigger
[SWS_CM_10515]	Silently discarding SOME/IP triggers for unsubscribed triggers
[SWS_CM_10519]	Content of the SOME/IP serialized trigger message
[SWS_CM_10520]	Content of the signal-based serialized trigger message
[SWS_CM_10521]	Checks for a received SOME/IP serialized trigger message
[SWS_CM_10522]	Checks for a received signal-based serialized trigger
[SWS_CM_11006]	Mapping of FindService method
[SWS_CM_11009]	Discovering remote Service Instances through DDS DomainParticipants
[SWS_CM_11041]	DDS serialization of `StdCppImplementationDataType` of `category` `VALUE`
[SWS_CM_11108]	Notifying the client of a response to a method call
[SWS_CM_11270]	Selecting elements of the ServiceInterface for SecOC transmission
[SWS_CM_11271]	SecOC secure channel behavior
[SWS_CM_11272]	Lifecycle management of FVM
[SWS_CM_11273]	Initialization of the FVM
[SWS_CM_11274]	SecOC secure channel sending
[SWS_CM_11275]	SecOC secure message build attempts
[SWS_CM_11276]	SecOC secure channel reception
[SWS_CM_11277]	SecOC secure message verification attempts
[SWS_CM_11278]	SecOC verification results
[SWS_CM_11279]	SecOc override the verification result
[SWS_CM_11280]	
[SWS_CM_11281]	
[SWS_CM_11282]	
[SWS_CM_11283]	
[SWS_CM_11284]	
[SWS_CM_11285]	
[SWS_CM_11286]	
[SWS_CM_11287]	
[SWS_CM_11288]	
[SWS_CM_11289]	
[SWS_CM_11290]	
[SWS_CM_11291]	
[SWS_CM_11292]	
[SWS_CM_11293]	
[SWS_CM_11295]	
[SWS_CM_11296]	

▽

△

Number	Heading
[SWS_CM_11297]	
[SWS_CM_11298]	
[SWS_CM_11299]	
[SWS_CM_11300]	
[SWS_CM_11301]	
[SWS_CM_11302]	
[SWS_CM_11303]	
[SWS_CM_11304]	
[SWS_CM_11305]	
[SWS_CM_11306]	
[SWS_CM_11307]	
[SWS_CM_11309]	
[SWS_CM_11310]	
[SWS_CM_11311]	
[SWS_CM_11312]	
[SWS_CM_11313]	
[SWS_CM_11314]	
[SWS_CM_11315]	
[SWS_CM_11316]	
[SWS_CM_11317]	
[SWS_CM_11318]	
[SWS_CM_11319]	
[SWS_CM_11320]	
[SWS_CM_11322]	
[SWS_CM_11323]	
[SWS_CM_11324]	
[SWS_CM_11325]	
[SWS_CM_11327]	
[SWS_CM_11328]	
[SWS_CM_11329]	
[SWS_CM_11330]	
[SWS_CM_11331]	
[SWS_CM_11332]	
[SWS_CM_11333]	
[SWS_CM_11334]	
[SWS_CM_11335]	
[SWS_CM_11336]	
[SWS_CM_11337]	
[SWS_CM_11342]	

▽

△

Number	Heading
[SWS_CM_11344]	
[SWS_CM_11345]	
[SWS_CM_11346]	
[SWS_CM_11350]	Execution Context for process service method invocation
[SWS_CM_11358]	Execution Context to update the event cache
[SWS_CM_12367]	
[SWS_CM_80023]	Source of an event message
[SWS_CM_80024]	Destination of an event message
[SWS_CM_80025]	Content of the SOME/IP serialized event message
[SWS_CM_80026]	Content of the signal-based serialized event message
[SWS_CM_80027]	Checks for a received SOME/IP serialized event message
[SWS_CM_80028]	Checks for a received signal-based serialized event message
[SWS_CM_80067]	Content of the SOME/IP serialized event message
[SWS_CM_80068]	Content of the signal-based serialized event message
[SWS_CM_80101]	Signal-based serialization
[SWS_CM_90101]	Secure UDP and TCP channel creation for TLS, DTLS and SecOC
[SWS_CM_90102]	Using secure TLS, DTLS and SecOC channels
[SWS_CM_90103]	TLS secure channel for ServiceInterface content using reliable transport
[SWS_CM_90104]	DTLS secure channel for ServiceInterface content using unreliable transport
[SWS_CM_90108]	SecOC secure channel for methods using reliable transport
[SWS_CM_90109]	SecOC secure channel for events and triggers using reliable transport
[SWS_CM_90110]	SecOC secure channel for fields
[SWS_CM_90111]	Behavior of a ServiceProxy over TLS before successful completion of the handshake
[SWS_CM_90112]	Behavior of a ServiceProxy over DTLS before successful completion of the handshake
[SWS_CM_90113]	Behavior of a ServiceSkeleton over TLS before successful completion of the handshake
[SWS_CM_90114]	Behavior of a ServiceSkeleton over DTLS before successful completion of the handshake
[SWS_CM_90115]	SecOC secure channel for methods using unreliable transport
[SWS_CM_90116]	SecOC secure channel for events and triggers using unreliable transport
[SWS_CM_90117]	IPsec secure channel between communication nodes
[SWS_CM_90118]	Transport of Service communication over an IPsec security association
[SWS_CM_90119]	Behavior of a creating ServiceProxy over TLS or DTLS
[SWS_CM_90121]	TLS server role of a Skeleton
[SWS_CM_90201]	Secure TLS and DTLS channel creation in the DDS Network Binding
[SWS_CM_90202]	Using TLS and DTLS secure channels in the DDS Network Binding
[SWS_CM_90203]	TLS secure channel for methods using reliable transport
[SWS_CM_90204]	DTLS secure channel for methods using unreliable transport

▽

△

Number	Heading
[SWS_CM_90205]	TLS secure channel for events using reliable transport
[SWS_CM_90206]	DTLS secure channel for events using unreliable transport
[SWS_CM_90207]	TLS secure channel for fields
[SWS_CM_90209]	IPsec secure channel between communication nodes and Transport of Service communication over an IPsec security association
[SWS_CM_90211]	Secure UDP and TCP channel creation for TLS and DTLS
[SWS_CM_90212]	Using secure TLS, DTLS channels
[SWS_CM_90213]	TLS secure channel for raw data streams using reliable transport
[SWS_CM_90214]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90215]	IPsec secure channel between communication nodes and Transport of Raw Data Stream communication over an IPsec security association
[SWS_CM_90437]	Send event where Communication Management is responsible for the data
[SWS_CM_90438]	Allocate data when Communication Management is responsible for the data
[SWS_CM_90477]	E2E Error Return Code
[SWS_CM_90481]	
[SWS_CM_90501]	Topic naming for Domain Participant USER_DATA QoS - based Service Instances
[SWS_CM_90510]	Mapping of FindService method
[SWS_CM_90513]	Discovering remote Service Instances through the ara.com://services/discovery topic
[SWS_CM_99024]	

Table E.12: Changed Specification Items in R22-11

E.3.3 Deleted Specification Items in R22-11

Number	Heading
[SWS_CM_10242]	Model representation of UTF-8 Strings
[SWS_CM_90424]	Provide E2E Result

Table E.13: Deleted Specification Items in R22-11

E.4 Constraint and Specification Item Changes between AUTOSAR Release R20-11 and R21-11

E.4.1 Added Specification Items in R21-11

Number	Heading
[SWS_CM_-CONSTR_00001]	
[SWS_CM_00035]	Re-entrancy and thread-safety - Unsubscribe
[SWS_CM_00104]	StopOfferService
[SWS_CM_00226]	Method to update the trigger counter
[SWS_CM_00227]	Sequence of actions in GetNewTriggers
[SWS_CM_00228]	Return Value
[SWS_CM_00249]	Enable service Trigger trigger
[SWS_CM_00351]	Trigger Receive Handler
[SWS_CM_00721]	Send trigger
[SWS_CM_00722]	Re-entrancy and thread-safety - Send
[SWS_CM_00723]	Method to subscribe to a service trigger
[SWS_CM_00724]	Re-entrancy and thread-safety - Subscribe
[SWS_CM_00810]	Method to unsubscribe from a service trigger
[SWS_CM_10445]	SomeIpBurstTransmission
[SWS_CM_10511]	Conditions for sending of a SOME/IP trigger
[SWS_CM_10512]	Content of the SOME/IP trigger
[SWS_CM_10513]	Checks for a received SOME/IP trigger
[SWS_CM_10514]	Identifying the right trigger
[SWS_CM_10515]	Silently discarding SOME/IP triggers for unsubscribed triggers
[SWS_CM_10516]	Invoke receive handler
[SWS_CM_10517]	Failures in sending a SOME/IP trigger
[SWS_CM_10518]	Conditions for sending of a trigger
[SWS_CM_10519]	Content of the SOME/IP serialized trigger message
[SWS_CM_10520]	Content of the signal-based serialized trigger message
[SWS_CM_10521]	Checks for a received SOME/IP serialized trigger message
[SWS_CM_10522]	Checks for a received signal-based serialized trigger
[SWS_CM_10523]	Silently discarding trigger for unsubscribed triggers
[SWS_CM_10524]	Mapping Triggers to DDS Topics
[SWS_CM_10525]	DDS Topic data type definition
[SWS_CM_10526]	Mapping of Send method
[SWS_CM_10527]	Mapping of Subscribe method
[SWS_CM_10528]	Creating a DDS DataReader for trigger subscription

△

Number	Heading
[SWS_CM_10529]	Defining a DDS DataReaderListener
[SWS_CM_10530]	Mapping of Unsubscribe method
[SWS_CM_10531]	Mapping of GetSubscriptionState method
[SWS_CM_10532]	Mapping of GetNewTriggers method
[SWS_CM_10534]	Mapping of SetReceiveHandler method
[SWS_CM_10535]	Mapping of UnsetReceiveHandler method
[SWS_CM_10536]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_10537]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_10538]	Restrictions on sending triggers
[SWS_CM_10539]	Restrictions on receiving triggers
[SWS_CM_10540]	Remote access control on providing triggers
[SWS_CM_10541]	Remote access control on consuming triggers
[SWS_CM_10550]	Assigning a DDS Topic and a DDS DataWriter to every Trigger in the ServiceInterface
[SWS_CM_11251]	Re-entrancy and thread-safety - GetNewTriggers
[SWS_CM_11370]	ServiceSkeleton destructor
[SWS_CM_11371]	HandleType destructor
[SWS_CM_12000]	Implementation types header files directory structure
[SWS_CM_12001]	C++ Implementation Data Types files
[SWS_CM_80501]	Mapping of Offer Service (Signal-Based Static network binding)
[SWS_CM_80502]	Mapping of Find Service (Signal-Based Static network binding)
[SWS_CM_80503]	Mapping of Subscribe Service (Signal-Based Static network binding)
[SWS_CM_80504]	Configuration of a data accumulation on a RequiredUserDefinedServiceInstance for transmission over UDP (Signal-Based Static network binding)
[SWS_CM_80505]	Data accumulation for UDP data transmission (Signal-Based Static network binding)
[SWS_CM_80506]	Arbitrary Message Header usage for Signal-Based Static network binding messages
[SWS_CM_80507]	No header option for Signal-Based Static network binding messages
[SWS_CM_80508]	No method support for Signal-Based Static network binding
[SWS_CM_80509]	Only field notifier support for Signal-Based Static network binding
[SWS_CM_80510]	Ignoring not mapped elements
[SWS_CM_80511]	Deserializing incomplete data belonging to a field
[SWS_CM_80512]	Mapping of Stop Offer Service (Signal-Based Static network binding)
[SWS_CM_80513]	Mapping of Unsubscribe Service (Signal-Based Static network binding)
[SWS_CM_90216]	Socket Options configuration
[SWS_CM_90217]	TLS properties configuration
[SWS_CM_90218]	Enforcement of IAM grants through DDS Security
[SWS_CM_90426]	Mapping of ProfileCheckStatus

▽

△

Number	Heading
[SWS_CM_90427]	Mapping of SMState
[SWS_CM_90500]	Choice of Service Instance discovery protocol
[SWS_CM_90501]	Topic naming for Domain Participant USER_DATA QoS - based Service Instances
[SWS_CM_90502]	Mapping of OfferService method
[SWS_CM_90503]	Assigning a DDS DomainParticipant to a Service Instance
[SWS_CM_90504]	Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface
[SWS_CM_90505]	Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Methods in the ServiceInterface
[SWS_CM_90506]	Assigning a DDS Topic and a DDS DataWriter to every Field in the ServiceInterface with its hasNotifier attribute equal to true
[SWS_CM_90507]	Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Field Getters/Setters in the ServiceInterface
[SWS_CM_90508]	Advertising Service IDs, Service Instance IDs, and ServiceInterface Contract Versions over the ara.com://services/discovery topic
[SWS_CM_90509]	Mapping of StopOfferService method
[SWS_CM_90510]	Mapping of FindService method
[SWS_CM_90511]	Finding a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_90512]	Creating a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_90513]	Discovering remote Service Instances through the ara.com://services/discovery topic
[SWS_CM_90514]	Mapping of StartFindService method
[SWS_CM_90515]	Mapping of StopFindService method
[SWS_CM_99028]	Types of APIs - Communication and Service Discovery APIs

Table E.14: Added Specification Items in R21-11

E.4.2 Changed Specification Items in R21-11

Number	Heading
[SWS_CM_00009]	Re-entrancy and thread-safety - General
[SWS_CM_00010]	Re-entrancy and thread-safety - OfferService
[SWS_CM_00011]	Re-entrancy and thread-safety- StopOfferService
[SWS_CM_00012]	Re-entrancy and thread-safety - Send
[SWS_CM_00013]	Re-entrancy and thread-safety - Allocate
[SWS_CM_00014]	Re-entrancy and thread-safety - RegisterGetHandler

▽

△

Number	Heading
[SWS_CM_00015]	Re-entrancy and thread-safety - RegisterSetHandler
[SWS_CM_00016]	Re-entrancy and thread-safety - Update
[SWS_CM_00017]	Re-entrancy and thread-safety - ServiceSkeleton method implementation
[SWS_CM_00018]	Re-entrancy and thread-safety - FindService
[SWS_CM_00019]	Re-entrancy and thread-safety - StartFindService
[SWS_CM_00020]	Re-entrancy and thread-safety - StopFindService
[SWS_CM_00021]	Re-entrancy and thread-safety - GetHandle
[SWS_CM_00022]	Re-entrancy and thread-safety - Subscribe
[SWS_CM_00023]	Re-entrancy and thread-safety - Unsubscribe
[SWS_CM_00024]	Re-entrancy and thread-safety - GetSubscriptionState
[SWS_CM_00025]	Re-entrancy and thread-safety - SetSubscriptionStateChangeHandler
[SWS_CM_00026]	Re-entrancy and thread-safety - UnsetSubscriptionStateChangeHandler
[SWS_CM_00027]	Re-entrancy and thread-safety - GetFreeSampleCount
[SWS_CM_00028]	Re-entrancy and thread-safety - SetReceiveHandler
[SWS_CM_00029]	Re-entrancy and thread-safety - UnsetReceiveHandler
[SWS_CM_00030]	Re-entrancy and thread-safety - Get
[SWS_CM_00031]	Re-entrancy and thread-safety - Set
[SWS_CM_00032]	Re-entrancy and thread-safety - Method call operator
[SWS_CM_00102]	Uniqueness of offered service on local machine
[SWS_CM_00103]	Protocol where a service is offered
[SWS_CM_00119]	Update Function
[SWS_CM_00141]	Method to subscribe to a service event
[SWS_CM_00162]	Send event where application is responsible for the data
[SWS_CM_00191]	Provision of method
[SWS_CM_00196]	Initiate a method call
[SWS_CM_00199]	Process Service method invocation
[SWS_CM_00209]	Start of service discovery protocol on Client side
[SWS_CM_00253]	Default size of length field for structs
[SWS_CM_00254]	Precedence when setting size of length field for structs
[SWS_CM_00255]	Default size of length field for structs
[SWS_CM_00256]	Default data type for the length field of structs
[SWS_CM_00258]	Default size of the length field for arrays
[SWS_CM_00259]	Setting size of the length field for arrays
[SWS_CM_00260]	Datatype for the length field of arrays
[SWS_CM_00264]	Setting the size of the length field for associative maps
[SWS_CM_00265]	Datatype for the length field of associative maps
[SWS_CM_00301]	Method Call Processing Mode
[SWS_CM_00302]	Instance Identifier Class

▽

△

Number	Heading
[SWS_CM_00306]	Sample Pointer
[SWS_CM_00310]	Subscription State
[SWS_CM_00701]	Method to update the event cache
[SWS_CM_00703]	Sequence of actions in GetNewSamples
[SWS_CM_00704]	Return Value
[SWS_CM_00705]	Query Free Sample Slots
[SWS_CM_00710]	No implicit context switches
[SWS_CM_00714]	Re-entrancy and thread-safety - GetNewSamples
[SWS_CM_01004]	Inclusion of common header file
[SWS_CM_01010]	Service Identifier and Service Contract Version
[SWS_CM_01020]	Common/Service header files directory structure
[SWS_CM_01069]	Variant function to create a new value in-place, in an existing Variant object by destoying and initializing the contained value using an initializer list
[SWS_CM_10258]	Default size of the length field for arrays
[SWS_CM_10267]	Insertion of an associative map length field
[SWS_CM_10269]	Setting the byte order of the length field for structs
[SWS_CM_10270]	Default byte order for the length field of structs
[SWS_CM_10273]	Size of length field for strings
[SWS_CM_10274]	Setting byte order for the length field of strings
[SWS_CM_10275]	Default size of length field for strings
[SWS_CM_10276]	Default byte order for the length field of strings
[SWS_CM_10278]	Data type of the length field for strings
[SWS_CM_10280]	Setting the byte order for size of length field for arrays
[SWS_CM_10281]	Byte order of length field for arrays
[SWS_CM_10283]	Setting the byte order for size of the length field for associative maps
[SWS_CM_10284]	Default byte order for size of the length field for associative maps
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10361]	Serializing Enumeration Data Type
[SWS_CM_10372]	Inclusion of Implementation Types header files
[SWS_CM_10389]	Configuration of a data accumulation on a ProvidedSomeipServiceInstance for transmission over UDP
[SWS_CM_10391]	Serializing Scale Linear And Texttable Data Type
[SWS_CM_10432]	
[SWS_CM_10451]	InstanceIdentifierContainer check during the creation of service skeleton
[SWS_CM_10458]	Handling of an ServiceInterface that does not contain any events, methods, or fields
[SWS_CM_10475]	
[SWS_CM_10476]	Defining a RawDataStream
[SWS_CM_10477]	Connect stream link

▽

△

Number	Heading
[SWS_CM_10482]	
[SWS_CM_10484]	
[SWS_CM_10485]	
[SWS_CM_10486]	
[SWS_CM_10487]	
[SWS_CM_11001]	Mapping of OfferService method
[SWS_CM_11005]	Mapping of StopOfferService method
[SWS_CM_11015]	Mapping Events to DDS Topics
[SWS_CM_11016]	DDS Topic data type definition
[SWS_CM_11019]	Creating a DDS DataReader for event subscription
[SWS_CM_11023]	Mapping of GetNewSamples method
[SWS_CM_11042]	DDS serialization of enumeration data types
[SWS_CM_11100]	Mapping Methods to DDS Service Methods and Topics
[SWS_CM_11102]	DDS Service Reply Topic data type definition
[SWS_CM_11103]	Creating a DataWriter to handle method requests on the client side
[SWS_CM_11104]	Creating a DataReader to handle method responses on the client side
[SWS_CM_11105]	Creating a DataReader to handle method requests on the server side
[SWS_CM_11106]	Creating a DataWriter to handle method responses on the server side
[SWS_CM_11130]	Mapping Fields with hasNotifier attribute to DDS Topics
[SWS_CM_11133]	Mapping of Subscribe method
[SWS_CM_11134]	Creating a DDS DataReader for field subscription
[SWS_CM_11144]	Mapping of Field Get/Set methods to DDS Service Methods and Topics
[SWS_CM_11147]	Creating a DataWriter to handle get/set requests on the client side
[SWS_CM_11148]	Creating a DataReader to handle get/set responses on the client side
[SWS_CM_11149]	Creating a DataReader to handle get/set requests on the server side
[SWS_CM_11150]	Creating a DataWriter to handle get/set responses on the server side
[SWS_CM_11262]	Missing alignment for a variable data length data element
[SWS_CM_11263]	Precedence of alignment settings for a variable data length data element
[SWS_CM_11286]	
[SWS_CM_11307]	
[SWS_CM_11309]	
[SWS_CM_11310]	
[SWS_CM_11312]	
[SWS_CM_11318]	
[SWS_CM_11319]	
[SWS_CM_11320]	
[SWS_CM_11322]	
[SWS_CM_11323]	

▽

△

Number	Heading
[SWS_CM_11324]	
[SWS_CM_11325]	
[SWS_CM_11345]	
[SWS_CM_11346]	
[SWS_CM_11350]	Execution Context for process service method invocation
[SWS_CM_11352]	Execution Context for finding service with handler registration using Instance ID
[SWS_CM_11354]	Execution Context for setting Subscription State change handler
[SWS_CM_11356]	Execution Context for enabling service event trigger
[SWS_CM_11358]	Execution Context to update the event cache
[SWS_CM_11360]	Execution Context for registering Getters
[SWS_CM_11362]	Execution Context for registering Setters
[SWS_CM_12367]	
[SWS_CM_80019]	Configuration of a data accumulation on a ProvidedSomeipServiceInstance for transmission over UDP
[SWS_CM_80027]	Checks for a received SOME/IP serialized event message
[SWS_CM_80028]	Checks for a received signal-based serialized event message
[SWS_CM_80103]	Deserializing incomplete data belonging to a field
[SWS_CM_90109]	SecOC secure channel for events and triggers using reliable transport
[SWS_CM_90113]	Behavior of a ServiceSkeleton over TLS before successful completion of the handshake
[SWS_CM_90114]	Behavior of a ServiceSkeleton over DTLS before successful completion of the handshake
[SWS_CM_90116]	SecOC secure channel for events and triggers using unreliable transport
[SWS_CM_90213]	TLS secure channel for raw data streams using reliable transport
[SWS_CM_90214]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90421]	ara::com::e2e::ProfileCheckStatus
[SWS_CM_90422]	ara::com::e2e::SMState
[SWS_CM_90431]	
[SWS_CM_90437]	Send event where Communication Management is responsible for the data
[SWS_CM_90443]	Wire type for non-dynamic data types
[SWS_CM_90444]	Wire type for dynamic data types
[SWS_CM_90445]	A deserializer shall always be able to handle the wire types 4, 5, 6 and 7
[SWS_CM_90446]	Data ID
[SWS_CM_90451]	Byte order for the length field of serialized structs
[SWS_CM_90452]	Default byte order for the length field of structs
[SWS_CM_90483]	
[SWS_CM_90484]	
[SWS_CM_90486]	
[SWS_CM_90487]	

▽

△

Number	Heading
[SWS_CM_90488]	
[SWS_CM_90489]	
[SWS_CM_90490]	
[SWS_CM_90491]	
[SWS_CM_90492]	
[SWS_CM_90493]	
[SWS_CM_90494]	
[SWS_CM_90495]	
[SWS_CM_90496]	
[SWS_CM_90497]	
[SWS_CM_90498]	
[SWS_CM_90499]	
[SWS_CM_99004]	Ethernet endpoint configuration
[SWS_CM_99005]	Wait for incoming connections
[SWS_CM_99006]	Timeout handling

Table E.15: Changed Specification Items in R21-11

E.4.3 Deleted Specification Items in R21-11

Number	Heading
[SWS_CM_00400]	Naming of data types by short name
[SWS_CM_00402]	Primitive fixed width integer types
[SWS_CM_00403]	StdCppImplementationDataType of category with one dimension
[SWS_CM_00404]	Array Data Type with more than one dimension
[SWS_CM_00405]	Structure Data Type
[SWS_CM_00406]	StdCppImplementationDataType with the category
[SWS_CM_00407]	StdCppImplementationDataType of Identifiable.category VECTOR with one dimension defined without an Allocator
[SWS_CM_00408]	Vector Data Type with more than one dimension
[SWS_CM_00409]	StdCppImplementationDataType with Identifiable.category ASSOCIATIVE_MAP defined without an Allocator
[SWS_CM_00410]	Data Type redefinition
[SWS_CM_00411]	Avoid Data Type redeclaration
[SWS_CM_00414]	Element specification typed by CppImplementationDataType
[SWS_CM_00421]	Provide data type definitions
[SWS_CM_00423]	Data Type Mapping
[SWS_CM_00424]	Enumeration Data Type

▽

△

Number	Heading
[SWS_CM_00425]	Definition of enumerators
[SWS_CM_00426]	Reject incomplete Enumeration Data Types
[SWS_CM_00449]	Variant Data Type
[SWS_CM_00450]	Define the maximum size of allocated vector memory
[SWS_CM_00452]	Usage of attribute CppImplementationDataType.arraySize of an CppImplementationDataType with category
[SWS_CM_00502]	CustomCppImplementationDataType of Identifiable.category
[SWS_CM_00503]	StdCppImplementationDataType of Identifiable.category VECTOR with one dimension defined with an Allocator
[SWS_CM_00504]	Supported Primitive Cpp Implementation Data Types
[SWS_CM_00505]	StdCppImplementationDataType with Identifiable.category ASSOCIATIVE_MAP defined with an Allocator
[SWS_CM_00506]	CustomCppImplementationDataType of category
[SWS_CM_00507]	CustomCppImplementationDataType of category
[SWS_CM_00508]	CustomCppImplementationDataType of Identifiable.category
[SWS_CM_00509]	StdCppImplementationDataType with the category with a defined Allocator
[SWS_CM_01032]	Accessing optional record elements inside a Structure Cpp Implementation Data Type that are serialized with the Tag-Length-Value principle.
[SWS_CM_10373]	Implementation Types header files existence
[SWS_CM_10374]	Data Type definitions for AUTOSAR Data Types in Implementation Types header files
[SWS_CM_10375]	Implementation Types header file namespace
[SWS_CM_10376]	Skip CompuScales with non-point range
[SWS_CM_10392]	ScaleLinearAndTexttable Class Template
[SWS_CM_10393]	ScaleLinearAndTexttable static assertion
[SWS_CM_10394]	ScaleLinearAndTexttable underlying type deduction
[SWS_CM_10395]	ScaleLinearAndTexttable default constructor
[SWS_CM_10396]	ScaleLinearAndTexttable copy constructor
[SWS_CM_10397]	ScaleLinearAndTexttable constructor with enum class argument
[SWS_CM_10398]	ScaleLinearAndTexttable constructor with underlying type argument
[SWS_CM_10399]	ScaleLinearAndTexttable copy assignment operator
[SWS_CM_10400]	ScaleLinearAndTexttable assignment operator with enum class argurment
[SWS_CM_10401]	ScaleLinearAndTexttable assignment operator with underlying type argument
[SWS_CM_10402]	ScaleLinearAndTexttable cast operator to the underlying type
[SWS_CM_10403]	Equal to operator between two ScaleLinearAndTexttable objects
[SWS_CM_10404]	Equal to operators between ScaleLinearAndTexttable and an underlying type
[SWS_CM_10405]	Equal to operators between ScaleLinearAndTexttable and an enum class
[SWS_CM_10406]	Not equal to operator between two ScaleLinearAndTexttable objects

▽

△

Number	Heading
[SWS_CM_10407]	Not equal to operators between ScaleLinearAndTexttable and an underlying type
[SWS_CM_10408]	Not equal to operators between ScaleLinearAndTexttable and an enum class
[SWS_CM_10409]	Scale Linear And Textable type definition
[SWS_CM_11004]	Adding Service and Service Instance IDs to the DDS DomainParticipant's USER_DATA QoS Policy
[SWS_CM_11308]	
[SWS_CM_11321]	
[SWS_CM_90210]	Using the DDS Security standard plug-ins in the Adaptive Platform

Table E.16: Deleted Specification Items in R21-11

E.5 Constraint and Specification Item Changes between AUTOSAR Release R19-11 and R20-11

E.5.1 Added Specification Items in R20-11

Number	Heading
[SWS_CM_00009]	Re-entrancy - General
[SWS_CM_00010]	Re-entrancy - OfferService
[SWS_CM_00011]	Re-entrancy - StopOfferService
[SWS_CM_00012]	Re-entrancy - Send
[SWS_CM_00013]	Re-entrancy - Allocate
[SWS_CM_00014]	Re-entrancy - RegisterGetHandler
[SWS_CM_00015]	Re-entrancy - RegisterSetHandler
[SWS_CM_00016]	Re-entrancy - Update
[SWS_CM_00017]	Re-entrancy - ServiceSkeleton method implementation
[SWS_CM_00018]	Re-entrancy - FindService
[SWS_CM_00019]	Re-entrancy - StartFindService
[SWS_CM_00020]	Re-entrancy - StopFindService
[SWS_CM_00021]	Re-entrancy - GetHandle
[SWS_CM_00022]	Re-entrancy - Subscribe
[SWS_CM_00023]	Re-entrancy - Unsubscribe
[SWS_CM_00024]	Re-entrancy - GetSubscriptionState
[SWS_CM_00025]	Re-entrancy - SetSubscriptionStateChangeHandler
[SWS_CM_00026]	Re-entrancy - UnsetSubscriptionStateChangeHandler
[SWS_CM_00027]	Re-entrancy - GetFreeSampleCount

▽

△

Number	Heading
[SWS_CM_00028]	Re-entrancy - SetReceiveHandler
[SWS_CM_00029]	Re-entrancy - UnsetReceiveHandler
[SWS_CM_00030]	Re-entrancy - Get
[SWS_CM_00031]	Re-entrancy - Set
[SWS_CM_00032]	Re-entrancy - Method call operator
[SWS_CM_10230]	
[SWS_CM_10240]	
[SWS_CM_10360]	Failures in sending a SOME/IP event message
[SWS_CM_10363]	Failures in sending a SOME/IP event message
[SWS_CM_10447]	Dealing with unmodelled ApApplicationErrors
[SWS_CM_10491]	Re-establishing service connection
[SWS_CM_10492]	IAM Module Instantiation
[SWS_CM_10493]	Local Access Control Activation
[SWS_CM_10494]	Remote Access Control Activation
[SWS_CM_10495]	TLS-based Authentication
[SWS_CM_10496]	IP and IPsec-based Authentication
[SWS_CM_10497]	Authentication Failure
[SWS_CM_10498]	Remote access control on executing methods
[SWS_CM_10499]	Remote access control on providing methods
[SWS_CM_10500]	Remote access control on providing events
[SWS_CM_10501]	Remote access control on consuming events
[SWS_CM_10502]	Remote access control on providing field notifiers
[SWS_CM_10503]	Remote access control on providing field setters
[SWS_CM_10504]	Remote access control on providing field getters
[SWS_CM_10505]	Remote access control on consuming field notifiers
[SWS_CM_10506]	Remote access control on calling field setters
[SWS_CM_10507]	Remote access control on calling field getters
[SWS_CM_11269]	Definition of serialization technology
[SWS_CM_11270]	Selecting elements of the ServiceInterface for SecOC transmission
[SWS_CM_11271]	SecOC secure channel behavior
[SWS_CM_11272]	Lifecycle management of FVM
[SWS_CM_11273]	Initialization of the FVM
[SWS_CM_11274]	SecOC secure channel sending
[SWS_CM_11275]	SecOC secure message build attempts
[SWS_CM_11276]	SecOC secure channel reception
[SWS_CM_11277]	SecOC secure message verification attempts
[SWS_CM_11278]	SecOC verification results
[SWS_CM_11279]	SecOc override the verification result
[SWS_CM_11286]	

▽

△

Number	Heading
[SWS_CM_11287]	
[SWS_CM_11288]	
[SWS_CM_11289]	
[SWS_CM_11290]	
[SWS_CM_11291]	
[SWS_CM_11292]	
[SWS_CM_11293]	
[SWS_CM_11295]	
[SWS_CM_11296]	
[SWS_CM_11297]	
[SWS_CM_11298]	
[SWS_CM_11299]	
[SWS_CM_11300]	
[SWS_CM_11301]	
[SWS_CM_11302]	
[SWS_CM_11303]	
[SWS_CM_11304]	
[SWS_CM_11305]	
[SWS_CM_11306]	
[SWS_CM_11307]	
[SWS_CM_11308]	
[SWS_CM_11309]	
[SWS_CM_11310]	
[SWS_CM_11311]	
[SWS_CM_11312]	
[SWS_CM_11313]	
[SWS_CM_11314]	
[SWS_CM_11315]	
[SWS_CM_11316]	
[SWS_CM_11317]	
[SWS_CM_11318]	
[SWS_CM_11319]	
[SWS_CM_11320]	
[SWS_CM_11321]	
[SWS_CM_11322]	
[SWS_CM_11323]	
[SWS_CM_11324]	
[SWS_CM_11325]	
[SWS_CM_11326]	Creation of an object using Named Constructor approach

▽

△

Number	Heading
[SWS_CM_11327]	
[SWS_CM_11328]	
[SWS_CM_11329]	
[SWS_CM_11330]	
[SWS_CM_11331]	
[SWS_CM_11332]	
[SWS_CM_11333]	
[SWS_CM_11334]	
[SWS_CM_11335]	
[SWS_CM_11336]	
[SWS_CM_11337]	
[SWS_CM_11340]	Definition general ara::com::secoc errors
[SWS_CM_11341]	SecOcFvm errors domain
[SWS_CM_11342]	
[SWS_CM_11344]	
[SWS_CM_11345]	
[SWS_CM_11346]	
[SWS_CM_11350]	Execution Context for process service method invocation
[SWS_CM_11351]	Error behaviour of provided Execution Context for process service method invocation
[SWS_CM_11352]	Execution Context for finding service with handler registration using Instance ID
[SWS_CM_11353]	Error behavior of provided Execution Context for finding service with handler registration using Instance ID
[SWS_CM_11354]	Execution Context for setting Subscription State change handler
[SWS_CM_11355]	Error behaviour of provided Execution Context for setting Subscription State change handler
[SWS_CM_11356]	Execution Context for enabling service event trigger
[SWS_CM_11357]	Error behaviour of provided Execution Context for enabling service event trigger
[SWS_CM_11358]	Execution Context to update the event cache
[SWS_CM_11359]	Error behaviour of provided Execution Context to update the event cache
[SWS_CM_11360]	Execution Context for registering Getters
[SWS_CM_11361]	Error behaviour of provided Execution Context for registering Getters
[SWS_CM_11362]	Execution Context for registering Setters
[SWS_CM_11363]	Error behaviour of provided Execution Context for registering Setters
[SWS_CM_11364]	Minimal behaviour of provided Execution Context
[SWS_CM_90453]	
[SWS_CM_90454]	
[SWS_CM_90455]	

▽

△

Number	Heading
[SWS_CM_90456]	
[SWS_CM_90457]	
[SWS_CM_90458]	
[SWS_CM_90459]	
[SWS_CM_90460]	
[SWS_CM_90461]	
[SWS_CM_90462]	
[SWS_CM_90463]	
[SWS_CM_90464]	E2E Error Handler - Invocation
[SWS_CM_90465]	E2E Error Handler - Invocation Arguments
[SWS_CM_90466]	Payload of the E2E Error Response
[SWS_CM_90467]	Payload of the Normal or Application Error Response
[SWS_CM_90468]	
[SWS_CM_90469]	
[SWS_CM_90470]	
[SWS_CM_90471]	
[SWS_CM_90472]	
[SWS_CM_90473]	
[SWS_CM_90474]	
[SWS_CM_90475]	
[SWS_CM_90476]	
[SWS_CM_90477]	E2E Error Return Code
[SWS_CM_90478]	
[SWS_CM_90479]	
[SWS_CM_90480]	
[SWS_CM_90481]	
[SWS_CM_90482]	
[SWS_CM_90483]	
[SWS_CM_90484]	
[SWS_CM_90485]	
[SWS_CM_90486]	
[SWS_CM_90487]	
[SWS_CM_90488]	
[SWS_CM_90489]	
[SWS_CM_90490]	
[SWS_CM_90491]	
[SWS_CM_90492]	
[SWS_CM_90493]	
[SWS_CM_90494]	

▽

△

Number	Heading
[SWS_CM_90495]	
[SWS_CM_90496]	
[SWS_CM_90497]	
[SWS_CM_90498]	
[SWS_CM_90499]	
[SWS_CM_99000]	CommunicationGroupServer Service
[SWS_CM_99001]	Broadcast method of CommunicationGroupServer Service
[SWS_CM_99002]	Peer To Peer Message method of CommunicationGroupServer Service
[SWS_CM_99004]	Attributes for the RawDataStream instance
[SWS_CM_99005]	Wait for incoming connections
[SWS_CM_99006]	Timeout handling
[SWS_CM_99007]	CommunicationGroupClient Service
[SWS_CM_99008]	Message method of CommunicationGroupClient Service
[SWS_CM_99009]	Message Response event of CommunicationGroupClient Service
[SWS_CM_99010]	Broadcast task
[SWS_CM_99011]	Peer To Peer message task
[SWS_CM_99012]	Message Response task
[SWS_CM_99013]	List Clients task
[SWS_CM_99014]	Message Response event of CommunicationGroupServer Service
[SWS_CM_99015]	List Clients method of CommunicationGroupServer Service
[SWS_CM_99016]	Connection Status of a Communication Group Server
[SWS_CM_99017]	Identifiable.category value COMMUNICATION_GROUP
[SWS_CM_99018]	Identifiable.category value COMMUNICATION_GROUP_SERVER
[SWS_CM_99019]	Identifiable.category value COMMUNICATION_GROUP_CLIENT
[SWS_CM_99020]	Communcation Group template
[SWS_CM_99021]	SHORT-NAME value of generated CommunicationGroupServer service
[SWS_CM_99022]	SHORT-NAME value of generated CommunicationGroupClient service
[SWS_CM_99023]	Definition general ara::com::cg errors
[SWS_CM_99024]	
[SWS_CM_99025]	Raw errors domain
[SWS_CM_99026]	E2E errors domain
[SWS_CM_99027]	Cg errors domain

Table E.17: Added Specification Items in R20-11

E.5.2 Changed Specification Items in R20-11

Number	Heading
[SWS_CM_00101]	Method to offer a service
[SWS_CM_00102]	Uniqueness of offered service on local machine
[SWS_CM_00114]	Registering Getters
[SWS_CM_00116]	Registering Setters
[SWS_CM_00118]	Method Instance Specifier Translation
[SWS_CM_00119]	Update Function
[SWS_CM_00122]	Find service with immediately returned request using Instance ID
[SWS_CM_00123]	Find service with handler registration using Instance ID
[SWS_CM_00128]	Ensuring the existence of valid Field values
[SWS_CM_00129]	Ensuring the existence of SetHandler
[SWS_CM_00141]	Method to subscribe to a service event
[SWS_CM_00151]	Method to unsubscribe from a service event
[SWS_CM_00152]	Creation of service skeleton using Instance Spec
[SWS_CM_00162]	Send event where application is responsible for the data
[SWS_CM_00181]	Enable service event trigger
[SWS_CM_00183]	Disable service event trigger
[SWS_CM_00202]	SOME/IP FindService message
[SWS_CM_00203]	SOME/IP OfferService message
[SWS_CM_00204]	SOME/IP StopOffer message
[SWS_CM_00205]	Content of SOME/IP SubscribeEventgroup message
[SWS_CM_00206]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_00207]	Content of SOME/IP StopSubscribeEventgroup message
[SWS_CM_00208]	SOME/IP SubscribeEventgroupNack message
[SWS_CM_00302]	Instance Identifier Class
[SWS_CM_00306]	Sample Pointer
[SWS_CM_00333]	Set Subscription State change handler
[SWS_CM_00403]	StdCppImplementationDataType of category with one dimension
[SWS_CM_00404]	Array Data Type with more than one dimension
[SWS_CM_00503]	StdCppImplementationDataType of Identifiable.category VECTOR with one dimension defined with an Allocator
[SWS_CM_00622]	Find service with immediately returned request using Instance Specifier
[SWS_CM_00623]	Find service with handler registration using Instance Specifier
[SWS_CM_00700]	Ensure memory allocation of maxSampleCount samples
[SWS_CM_00704]	Return Value
[SWS_CM_00707]	Calculation of Free Sample Count
[SWS_CM_01010]	Service Identifier, Service Version Classes and Service Contract Version

▽

△

Number	Heading
[SWS_CM_01059]	Variant destructor
[SWS_CM_10247]	Deserialization of strings
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10308]	Conditions for sending of a SOME/IP response message
[SWS_CM_10312]	Content of the SOME/IP response message
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10344]	Content of the SOME/IP response message
[SWS_CM_10357]	Distinguishing errors from normal responses
[SWS_CM_10358]	Identifying the right application error in a message with Message Type set to RESPONSE (0x80)
[SWS_CM_10410]	InstanceIdentifier check during the creation of service skeleton
[SWS_CM_10429]	Identifying the right application error in a message with Message Type set to ERROR (0x81)
[SWS_CM_10430]	Handling invalid messages with Message Type set to ERROR (0x81)
[SWS_CM_10431]	Mapping of ara::core::ErrorCode
[SWS_CM_10432]	
[SWS_CM_10435]	Exception-less creation of service skeleton using Instance ID
[SWS_CM_10436]	Exception-less creation of service skeleton using Instance Spec
[SWS_CM_10437]	Exception-less creation of service skeleton using Instance ID Container
[SWS_CM_10438]	Exception-less creation of service proxy
[SWS_CM_10450]	InstanceSpecifier check during the creation of service skeleton
[SWS_CM_10453]	Implementation of SwDataDefProps.invalidValue
[SWS_CM_10462]	
[SWS_CM_10463]	
[SWS_CM_10464]	
[SWS_CM_10465]	
[SWS_CM_10467]	
[SWS_CM_10468]	
[SWS_CM_10469]	
[SWS_CM_10470]	E2E Error Handler - Existence
[SWS_CM_10471]	E2E Error Handler - Invocation Arguments
[SWS_CM_10472]	E2E Error Response
[SWS_CM_10473]	Handling the E2E Error Response
[SWS_CM_10474]	

▽

△

Number	Heading
[SWS_CM_10475]	
[SWS_CM_10476]	Defining a RawDataStream
[SWS_CM_10477]	Connect stream link
[SWS_CM_10478]	Shutdown stream link
[SWS_CM_10479]	Read data from stream
[SWS_CM_10480]	Write data to stream
[SWS_CM_10481]	
[SWS_CM_10482]	
[SWS_CM_10483]	
[SWS_CM_10484]	
[SWS_CM_10485]	
[SWS_CM_10486]	
[SWS_CM_10487]	
[SWS_CM_11001]	Mapping of OfferService method
[SWS_CM_11009]	Discovering remote Service Instances through DDS DomainParticipants
[SWS_CM_11016]	DDS Topic data type definition
[SWS_CM_11100]	Mapping Methods to DDS Service Methods and Topics
[SWS_CM_11103]	Creating a DataWriter to handle method requests on the client side
[SWS_CM_11104]	Creating a DataReader to handle method responses on the client side
[SWS_CM_11105]	Creating a DataReader to handle method requests on the server side
[SWS_CM_11106]	Creating a DataWriter to handle method responses on the server side
[SWS_CM_11131]	Field Notifier DDS Topic data type definition
[SWS_CM_11144]	Mapping of Field Get/Set methods to DDS Service Methods and Topics
[SWS_CM_11268]	Definition general ara::com::raw errors
[SWS_CM_12367]	
[SWS_CM_80021]	Conditions for sending of an event message
[SWS_CM_80023]	Source of an event message
[SWS_CM_80024]	Destination of an event message
[SWS_CM_80025]	Content of the SOME/IP serialized event message
[SWS_CM_80027]	Checks for a received SOME/IP serialized event message
[SWS_CM_80030]	Silently discarding event messages for unsubscribed events
[SWS_CM_80032]	Deserializing the SOME/IP serialized payload
[SWS_CM_80033]	Deserializing the signal-based serialized payload
[SWS_CM_80063]	Conditions for sending of an event message
[SWS_CM_80067]	Content of the SOME/IP serialized event message
[SWS_CM_80072]	Silently discarding event messages for unsubscribed events
[SWS_CM_80074]	Deserializing the SOME/IP serialized payload
[SWS_CM_80075]	Deserializing the signal-based payload

▽

△

Number	Heading
[SWS_CM_90001]	Restrictions on executing methods
[SWS_CM_90002]	Restrictions on sending events
[SWS_CM_90003]	Restrictions on receiving events
[SWS_CM_90005]	Restrictions on offering services
[SWS_CM_90006]	Restrictions on using services
[SWS_CM_90007]	Restrictions on using RawDataStreams
[SWS_CM_90102]	Using secure TLS, DTLS and SecOC channels
[SWS_CM_90103]	TLS secure channel for ServiceInterface content using reliable transport
[SWS_CM_90104]	DTLS secure channel for ServiceInterface content using unreliable transport
[SWS_CM_90111]	Behavior of a ServiceProxy over TLS before successful completion of the handshake
[SWS_CM_90115]	SecOC secure channel for methods using unreliable transport
[SWS_CM_90121]	TLS server role of a Skeleton
[SWS_CM_90203]	TLS secure channel for methods using reliable transport
[SWS_CM_90204]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90401]	
[SWS_CM_90403]	
[SWS_CM_90404]	
[SWS_CM_90408]	
[SWS_CM_90410]	
[SWS_CM_90411]	
[SWS_CM_90412]	
[SWS_CM_90413]	
[SWS_CM_90415]	
[SWS_CM_90416]	
[SWS_CM_90417]	
[SWS_CM_90421]	ara::com::e2e::ProfileCheckStatus
[SWS_CM_90422]	ara::com::e2e::SMState
[SWS_CM_90430]	
[SWS_CM_90431]	
[SWS_CM_90433]	
[SWS_CM_90437]	Send event where Communication Management is responsible for the data
[SWS_CM_90438]	Allocating data for event transfer

Table E.18: Changed Specification Items in R20-11

E.5.3 Deleted Specification Items in R20-11

Number	Heading
[SWS_CM_00350]	Instance Specifier Class
[SWS_CM_10433]	Declaration of Construction Token
[SWS_CM_10434]	Creation of a Construction Token
[SWS_CM_10461]	
[SWS_CM_80002]	
[SWS_CM_80005]	Start of service discovery protocol on Server side
[SWS_CM_80006]	Start of service discovery protocol on Client side
[SWS_CM_80007]	SOME/IP FindService message
[SWS_CM_80008]	SOME/IP OfferService message
[SWS_CM_80009]	SOME/IP StopOffer message
[SWS_CM_80010]	Sending SOME/IP SubscribeEventgroup messages - initial
[SWS_CM_80011]	Sending SOME/IP SubscribeEventgroup messages - renewal
[SWS_CM_80012]	Content of SOME/IP SubscribeEventgroup message
[SWS_CM_80013]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_80014]	SOME/IP SubscribeEventgroupNack message
[SWS_CM_80015]	Sending SOME/IP StopSubscribeEventgroup messages
[SWS_CM_80016]	Content of SOME/IP StopSubscribeEventgroup message
[SWS_CM_80018]	Enabling of data accumulation for UDP data transmission
[SWS_CM_80029]	Identifying the right event
[SWS_CM_80031]	Invoke receive handler
[SWS_CM_80034]	Providing the received event data
[SWS_CM_80035]	Conditions for sending of a SOME/IP request message
[SWS_CM_80036]	Failures in sending of a SOME/IP request message
[SWS_CM_80037]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_80038]	Source of a SOME/IP request message
[SWS_CM_80039]	Destination of a SOME/IP request message
[SWS_CM_80040]	Content of the SOME/IP request message
[SWS_CM_80041]	Checks for a received SOME/IP request message
[SWS_CM_80042]	Identifying the right method
[SWS_CM_80043]	Deserializing the payload
[SWS_CM_80044]	Invoke the method - event driven
[SWS_CM_80045]	Invoke the method - polling
[SWS_CM_80046]	Conditions for sending of a SOME/IP response message
[SWS_CM_80047]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_80048]	Source of a SOME/IP response message
[SWS_CM_80049]	Destination of a SOME/IP response message

△

Number	Heading
[SWS_CM_80050]	Content of the SOME/IP response message
[SWS_CM_80051]	payload representing application error
[SWS_CM_80052]	Checks for a received SOME/IP response message
[SWS_CM_80053]	Identifying the right method
[SWS_CM_80054]	Discarding orphaned responses
[SWS_CM_80055]	Distinguishing errors from normal responses
[SWS_CM_80056]	Deserializing the payload - normal response messages
[SWS_CM_80057]	Failures during deserialization of response messages
[SWS_CM_80058]	Identifying the right application error in a message with Message Type set to RESPONSE (0x80)
[SWS_CM_80059]	Identifying the right application error in a message with Message Type set to ERROR (0x81)
[SWS_CM_80060]	Handling invalid messages with Message Type set to RESPONSE (0x81)
[SWS_CM_80061]	Making the Future ready
[SWS_CM_80062]	Invoke the notification function
[SWS_CM_80071]	Identifying the right event
[SWS_CM_80073]	Invoke receive handler
[SWS_CM_80076]	Providing the received event data
[SWS_CM_80077]	Conditions for sending of a SOME/IP request message
[SWS_CM_80078]	Failures in sending of a SOME/IP request message
[SWS_CM_80079]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_80080]	Source of a SOME/IP request message
[SWS_CM_80081]	Destination of a SOME/IP request message
[SWS_CM_80082]	Content of the SOME/IP request message
[SWS_CM_80083]	Checks for a received SOME/IP request message
[SWS_CM_80084]	Identifying the right method
[SWS_CM_80085]	Deserializing the payload
[SWS_CM_80086]	Invoke the registered set/get handlers - event driven
[SWS_CM_80087]	Invoke the registered set/get handlers - polling
[SWS_CM_80088]	Conditions for sending of a SOME/IP response message
[SWS_CM_80089]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_80090]	Source of a SOME/IP response message
[SWS_CM_80091]	Destination of a SOME/IP response message
[SWS_CM_80092]	Content of the SOME/IP response message
[SWS_CM_80093]	Checks for a received SOME/IP response message
[SWS_CM_80094]	Identifying the right method
[SWS_CM_80095]	Discarding orphaned responses
[SWS_CM_80096]	Deserializing the payload
[SWS_CM_80097]	Failures during deserialization of response messages

▽

△

Number	Heading
[SWS_CM_80098]	Making the Future ready
[SWS_CM_80099]	Invoke the notification function
[SWS_CM_90004]	Process separation of network and language binding for access control
[SWS_CM_90105]	TLS secure channel for events using reliable transport
[SWS_CM_90106]	DTLS secure channel for events using unreliable transport
[SWS_CM_90107]	TLS secure channel for fields
[SWS_CM_90120]	TLS client role of a Proxy
[SWS_CM_90405]	

Table E.19: Deleted Specification Items in R20-11

E.6 Constraint and Specification Item Changes between AUTOSAR Release R19-03 and R19-11

E.6.1 Added Specification Items in R19-11

Number	Heading
[SWS_CM_00700]	Ensure memory allocation of maxSampleCount samples
[SWS_CM_00701]	Method to update the event cache
[SWS_CM_00702]	Signature of Callable f
[SWS_CM_00703]	Sequence of actions in GetNewSamples
[SWS_CM_00704]	Return Value
[SWS_CM_00705]	Query Free Sample Slots
[SWS_CM_00706]	Return Value of GetFreeSampleCount
[SWS_CM_00707]	Calculation of Free Sample Count
[SWS_CM_00709]	FIFO semantics
[SWS_CM_00710]	No implicit context switches
[SWS_CM_00711]	
[SWS_CM_00714]	Reentrancy
[SWS_CM_09004]	Adding Service IDs, Service Instance IDs, and ServiceInterface Contract Versions to the DDS DomainParticipant's USER_DATA QoS Policy
[SWS_CM_10202]	Version blacklist
[SWS_CM_10416]	Reception of a malformed message
[SWS_CM_10440]	Aborting method calls in case of locally detected failures
[SWS_CM_10441]	Failures in sending of a SOME/IP request message
[SWS_CM_10442]	Failures during deserialization of response messages
[SWS_CM_10443]	Failures in sending of a SOME/IP request message

▽

△

Number	Heading
[SWS_CM_10444]	Failures during deserialization of response messages
[SWS_CM_10446]	Destruction of service proxy
[SWS_CM_10453]	Implementation of `invalidValue`
[SWS_CM_10454]	
[SWS_CM_10455]	
[SWS_CM_10456]	
[SWS_CM_10457]	
[SWS_CM_10458]	Handling of an ServiceInterface that does not contain any events, methods, or fields
[SWS_CM_10459]	
[SWS_CM_10460]	
[SWS_CM_10461]	
[SWS_CM_10462]	
[SWS_CM_10463]	
[SWS_CM_10464]	
[SWS_CM_10465]	
[SWS_CM_10466]	
[SWS_CM_10467]	
[SWS_CM_10468]	
[SWS_CM_10469]	
[SWS_CM_10470]	
[SWS_CM_10471]	E2E Error Handler
[SWS_CM_10472]	E2E Error Response
[SWS_CM_10473]	E2E Error Response
[SWS_CM_10475]	
[SWS_CM_10476]	Defining a RawDataStream
[SWS_CM_10477]	Connect stream link
[SWS_CM_10478]	Shutdown stream link
[SWS_CM_10479]	Read data from stream
[SWS_CM_10480]	Write data to stream
[SWS_CM_10481]	Class RawDataStream
[SWS_CM_10482]	RawDataStream Constructor
[SWS_CM_10483]	RawDataStream Destructor
[SWS_CM_10484]	Method Connect
[SWS_CM_10485]	Method Shutdown
[SWS_CM_10486]	Method ReadData
[SWS_CM_10487]	Method WriteData
[SWS_CM_10488]	Raw data stream header file existence
[SWS_CM_10489]	Raw data stream header file namespace

▽

△

Number	Heading
[SWS_CM_10490]	Data Type declarations in Raw data stream header file
[SWS_CM_11267]	General errors domain
[SWS_CM_11268]	Definition general ara::com::raw errors
[SWS_CM_12367]	
[SWS_CM_80001]	
[SWS_CM_80002]	
[SWS_CM_80003]	Byte order for signal-based network binding with SOME/IP serialization
[SWS_CM_80004]	Byte order for signal-based network binding with signal-based serialization
[SWS_CM_80005]	Start of service discovery protocol on Server side
[SWS_CM_80006]	Start of service discovery protocol on Client side
[SWS_CM_80007]	SOME/IP FindService message
[SWS_CM_80008]	SOME/IP OfferService message
[SWS_CM_80009]	SOME/IP StopOffer message
[SWS_CM_80010]	Sending SOME/IP SubscribeEventgroup messages - initial
[SWS_CM_80011]	Sending SOME/IP SubscribeEventgroup messages - renewal
[SWS_CM_80012]	Content of SOME/IP SubscribeEventgroup message
[SWS_CM_80013]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_80014]	SOME/IP SubscribeEventgroupNack message
[SWS_CM_80015]	Sending SOME/IP StopSubscribeEventgroup messages
[SWS_CM_80016]	Content of SOME/IP StopSubscribeEventgroup message
[SWS_CM_80017]	Data accumulation for UDP data transmission
[SWS_CM_80018]	Enabling of data accumulation for UDP data transmission
[SWS_CM_80019]	Configuration of a data accumulation on a `ProvidedServiceInstance` for transmission over UDP
[SWS_CM_80020]	Configuration of a data accumulation on a `RequiredSomeipServiceInstance` for transmission over UDP
[SWS_CM_80021]	Conditions for sending of an event message
[SWS_CM_80022]	Transport protocol for sending of an event message
[SWS_CM_80023]	Source of an event message
[SWS_CM_80024]	Destination of an event message
[SWS_CM_80025]	Content of the SOME/IP serialized event message
[SWS_CM_80026]	Content of the signal-based serialized event message
[SWS_CM_80027]	Checks for a received SOME/IP serialized event message
[SWS_CM_80028]	Checks for a received signal-based serialized event message
[SWS_CM_80029]	Identifying the right event
[SWS_CM_80030]	Silently discarding event messages for unsubscribed events
[SWS_CM_80031]	Invoke receive handler
[SWS_CM_80032]	Deserializing the SOME/IP serialized payload
[SWS_CM_80033]	Deserializing the signal-based serialized payload

▽

△

Number	Heading
[SWS_CM_80034]	Providing the received event data
[SWS_CM_80035]	Conditions for sending of a SOME/IP request message
[SWS_CM_80036]	Failures in sending of a SOME/IP request message
[SWS_CM_80037]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_80038]	Source of a SOME/IP request message
[SWS_CM_80039]	Destination of a SOME/IP request message
[SWS_CM_80040]	Content of the SOME/IP request message
[SWS_CM_80041]	Checks for a received SOME/IP request message
[SWS_CM_80042]	Identifying the right method
[SWS_CM_80043]	Deserializing the payload
[SWS_CM_80044]	Invoke the method - event driven
[SWS_CM_80045]	Invoke the method - polling
[SWS_CM_80046]	Conditions for sending of a SOME/IP response message
[SWS_CM_80047]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_80048]	Source of a SOME/IP response message
[SWS_CM_80049]	Destination of a SOME/IP response message
[SWS_CM_80050]	Content of the SOME/IP response message
[SWS_CM_80051]	payload representing application error
[SWS_CM_80052]	Checks for a received SOME/IP response message
[SWS_CM_80053]	Identifying the right method
[SWS_CM_80054]	Discarding orphaned responses
[SWS_CM_80055]	Distinguishing errors from normal responses
[SWS_CM_80056]	Deserializing the payload - normal response messages
[SWS_CM_80057]	Failures during deserialization of response messages
[SWS_CM_80058]	Identifying the right application error in a message with Message Type set to RESPONSE (0x80)
[SWS_CM_80059]	Identifying the right application error in a message with Message Type set to ERROR (0x81)
[SWS_CM_80060]	Handling invalid messages with Message Type set to RESPONSE (0x81)
[SWS_CM_80061]	Making the Future ready
[SWS_CM_80062]	Invoke the notification function
[SWS_CM_80063]	Conditions for sending of an event message
[SWS_CM_80064]	Transport protocol for sending of an event message
[SWS_CM_80065]	Source of an event message
[SWS_CM_80066]	Destination of an event message
[SWS_CM_80067]	Content of the SOME/IP serialized event message
[SWS_CM_80068]	Content of the signal-based serialized event message
[SWS_CM_80069]	Checks for a received SOME/IP serialized event message
[SWS_CM_80070]	Checks for a received signal-based event message

▽

△

Number	Heading
[SWS_CM_80071]	Identifying the right event
[SWS_CM_80072]	Silently discarding event messages for unsubscribed events
[SWS_CM_80073]	Invoke receive handler
[SWS_CM_80074]	Deserializing the SOME/IP serialized payload
[SWS_CM_80075]	Deserializing the signal-based payload
[SWS_CM_80076]	Providing the received event data
[SWS_CM_80077]	Conditions for sending of a SOME/IP request message
[SWS_CM_80078]	Failures in sending of a SOME/IP request message
[SWS_CM_80079]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_80080]	Source of a SOME/IP request message
[SWS_CM_80081]	Destination of a SOME/IP request message
[SWS_CM_80082]	Content of the SOME/IP request message
[SWS_CM_80083]	Checks for a received SOME/IP request message
[SWS_CM_80084]	Identifying the right method
[SWS_CM_80085]	Deserializing the payload
[SWS_CM_80086]	Invoke the registered set/get handlers - event driven
[SWS_CM_80087]	Invoke the registered set/get handlers - polling
[SWS_CM_80088]	Conditions for sending of a SOME/IP response message
[SWS_CM_80089]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_80090]	Source of a SOME/IP response message
[SWS_CM_80091]	Destination of a SOME/IP response message
[SWS_CM_80092]	Content of the SOME/IP response message
[SWS_CM_80093]	Checks for a received SOME/IP response message
[SWS_CM_80094]	Identifying the right method
[SWS_CM_80095]	Discarding orphaned responses
[SWS_CM_80096]	Deserializing the payload
[SWS_CM_80097]	Failures during deserialization of response messages
[SWS_CM_80098]	Making the Future ready
[SWS_CM_80099]	Invoke the notification function
[SWS_CM_80100]	SOME/IP serialization of signal-based network binding
[SWS_CM_80101]	`ServiceInstanceToSignalMapping` input for serialization of signal-based network binding
[SWS_CM_80102]	Ignoring not mapped elements
[SWS_CM_80103]	Init value for field elements
[SWS_CM_90007]	Restrictions on using RawDataStreams
[SWS_CM_90211]	Secure UDP and TCP channel creation for TLS and DTLS
[SWS_CM_90212]	Using secure TLS, DTLS channels
[SWS_CM_90213]	TLS secure channel for raw data streams using reliable transport
[SWS_CM_90214]	DTLS secure channel for methods using unreliable transport

▽

△

Number	Heading
[SWS_CM_90215]	IPsec secure channel between communication nodes and Transport of Raw Data Stream communication over an IPsec security association
[SWS_CM_99003]	

Table E.20: Added Specification Items in R19-11

E.6.2 Changed Specification Items in R19-11

Number	Heading
[SWS_CM_00002]	Service skeleton class
[SWS_CM_00003]	Service skeleton Event class
[SWS_CM_00004]	Service proxy class
[SWS_CM_00005]	Service proxy Event class
[SWS_CM_00006]	Service proxy Method class
[SWS_CM_00007]	Service skeleton Field class
[SWS_CM_00008]	Service proxy Field class
[SWS_CM_00101]	Method to offer a service
[SWS_CM_00111]	Method to stop offering a service
[SWS_CM_00112]	Method to get the value of a field
[SWS_CM_00113]	Method to set the value of a field
[SWS_CM_00114]	Registering Getters
[SWS_CM_00115]	Existence of RegisterGetHandler method
[SWS_CM_00116]	Registering Setters
[SWS_CM_00117]	Existence of the RegisterSetHandler method
[SWS_CM_00118]	Method Instance Specifier Translation
[SWS_CM_00119]	Update Function
[SWS_CM_00120]	Provision of an update notification event for a Field
[SWS_CM_00122]	Find service with immediately returned request using Instance ID
[SWS_CM_00123]	Find service with handler registration using Instance ID
[SWS_CM_00124]	Find service handler invocation
[SWS_CM_00125]	Stop find service
[SWS_CM_00130]	Creation of service skeleton using Instance ID
[SWS_CM_00131]	Creation of service proxy
[SWS_CM_00132]	Existence of getter method
[SWS_CM_00133]	Existence of the set method
[SWS_CM_00134]	Copy semantics of service skeleton class
[SWS_CM_00135]	Move semantics of service skeleton class

▽

△

Number	Heading
[SWS_CM_00136]	Copy semantics of service proxy class
[SWS_CM_00137]	Move semantics of service proxy class
[SWS_CM_00141]	Method to subscribe to a service event
[SWS_CM_00151]	Method to unsubscribe from a service event
[SWS_CM_00152]	Creation of service skeleton using Instance Spec
[SWS_CM_00153]	Creation of service skeleton using Instance ID Container
[SWS_CM_00162]	Send event where application is responsible for the data
[SWS_CM_00181]	Enable service event trigger
[SWS_CM_00183]	Disable service event trigger
[SWS_CM_00191]	Provision of method
[SWS_CM_00192]	Synchronous behavior of method call
[SWS_CM_00193]	Asynchronous behavior of method call with polling
[SWS_CM_00194]	Cancel the method call
[SWS_CM_00195]	Retrieving results of the method call
[SWS_CM_00196]	Initiate a method call
[SWS_CM_00197]	Asynchronous behavior of method call with notification
[SWS_CM_00198]	Set service method processing mode
[SWS_CM_00199]	Process Service method invocation
[SWS_CM_00202]	SOME/IP FindService message
[SWS_CM_00203]	SOME/IP OfferService message
[SWS_CM_00205]	Content of SOME/IP SubscribeEventgroup message
[SWS_CM_00206]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_00209]	Start of service discovery protocol on Client side
[SWS_CM_00301]	Method Call Processing Mode
[SWS_CM_00302]	Instance Identifier Class
[SWS_CM_00303]	Find Service Handle
[SWS_CM_00304]	Service Handle Container
[SWS_CM_00306]	Sample Pointer
[SWS_CM_00308]	Sample Allocatee Pointer
[SWS_CM_00309]	Event Receive Handler
[SWS_CM_00310]	Subscription State
[SWS_CM_00311]	Subscription State Changed Handler
[SWS_CM_00312]	Handle Type Class
[SWS_CM_00313]	Call SubscriptionStateChangeHandler with kSubscriptionPending
[SWS_CM_00314]	Call SubscriptionStateChangeHandler with kSubscribed
[SWS_CM_00315]	Re-establishing an active subscription
[SWS_CM_00316]	Query Subscription State
[SWS_CM_00317]	Copy semantics of handle Type Class

▽

△

Number	Heading
[SWS_CM_00318]	Move semantics of handle Type Class
[SWS_CM_00319]	Instance Identifier Container Class
[SWS_CM_00333]	Set Subscription State change handler
[SWS_CM_00334]	Unset Subscription State change handler
[SWS_CM_00350]	Instance Specifier Class
[SWS_CM_00383]	Find Service Handler
[SWS_CM_00402]	Primitive fixed width integer types
[SWS_CM_00403]	StdCppImplementationDataType of category ARRAY with one dimension
[SWS_CM_00404]	Array Data Type with more than one dimension
[SWS_CM_00405]	Structure Data Type
[SWS_CM_00406]	StdCppImplementationDataType with the category STRING
[SWS_CM_00407]	StdCppImplementationDataType of category VECTOR with one dimension defined without an Allocator
[SWS_CM_00409]	StdCppImplementationDataType with category ASSOCIATIVE_MAP defined without an Allocator
[SWS_CM_00410]	Data Type redefinition
[SWS_CM_00414]	Element specification typed by CppImplementationDataType
[SWS_CM_00424]	Enumeration Data Type
[SWS_CM_00425]	Definition of enumerators
[SWS_CM_00449]	Variant Data Type
[SWS_CM_00502]	CustomCppImplementationDataType of category ARRAY
[SWS_CM_00503]	StdCppImplementationDataType of category VECTOR with one dimension defined with an Allocator
[SWS_CM_00504]	Supported Primitive Cpp Implementation Data Types
[SWS_CM_00505]	StdCppImplementationDataType with category ASSOCIATIVE_MAP defined with an Allocator
[SWS_CM_00506]	CustomCppImplementationDataType of category ASSOCIATIVE_MAP
[SWS_CM_00507]	CustomCppImplementationDataType of category VECTOR
[SWS_CM_00508]	CustomCppImplementationDataType of category VARIANT
[SWS_CM_00509]	StdCppImplementationDataType with the category STRING with a defined Allocator
[SWS_CM_00622]	Find service with immediately returned request using Instance Specifier
[SWS_CM_00623]	Find service with handler registration using Instance Specifier
[SWS_CM_01001]	Inclusion of Types header file
[SWS_CM_01002]	Service header files existence
[SWS_CM_01004]	Inclusion of common header file
[SWS_CM_01005]	Namespace of Service header files
[SWS_CM_01006]	Service skeleton namespace
[SWS_CM_01007]	Service proxy namespace
[SWS_CM_01009]	Service events namespace

▽

△

Number	Heading
[SWS_CM_01010]	Service Identifier, Service Version Classes and Service Contract Version
[SWS_CM_01012]	Common header file existence
[SWS_CM_01013]	Types header file existence
[SWS_CM_01015]	Service methods namespace
[SWS_CM_01018]	Types header file namespace
[SWS_CM_01019]	Data Type declarations in Types header file
[SWS_CM_01020]	Folder structure
[SWS_CM_01031]	Service fields namespace
[SWS_CM_01032]	Accessing optional record elements inside a `Structure Cpp Implementation Data Type` that are serialized with the Tag-Length-Value principle.
[SWS_CM_01046]	Definition of `tlvDataIdDefinition`
[SWS_CM_01050]	`Variant` Class Template
[SWS_CM_01051]	`Variant` default constructor
[SWS_CM_01052]	`Variant` move constructor
[SWS_CM_01053]	`Variant` copy constructor
[SWS_CM_01054]	`Variant` converting constructor
[SWS_CM_01055]	`Variant` explicit converting constructor with specified alternative
[SWS_CM_01056]	`Variant` explicit converting constructor with specified alternative and initializer list
[SWS_CM_01057]	`Variant` explicit converting constructor with alternative specified by index
[SWS_CM_01058]	`Variant` explicit converting constructor with alternative specified by index and initializer list
[SWS_CM_01059]	`Variant` destructor
[SWS_CM_01060]	`Variant` move assignment operator
[SWS_CM_01061]	`Variant` default copy assignment operator
[SWS_CM_01062]	`Variant` converting assignment operator
[SWS_CM_01063]	`Variant` function to return the zero-based index of the alternative
[SWS_CM_01064]	`Variant` function to check if the Variant is in invalid state
[SWS_CM_01065]	`Variant` function to swap two Variants
[SWS_CM_01066]	`Variant` function to create a new value in-place, in an existing Variant object
[SWS_CM_01067]	`Variant` function to create a new value in-place, in an existing Variant object using an initializer list
[SWS_CM_01068]	`Variant` function to create a new value in-place, in an existing Variant object by destoying and initializing the contained value
[SWS_CM_01069]	`Variant` function to create a new value in-place, in an existing Variant object by destoying and initializing the contained value using an initializer list
[SWS_CM_10017]	
[SWS_CM_10054]	
[SWS_CM_10242]	Model representation of UTF-8 Strings
[SWS_CM_10245]	Serialization of strings

▽

△

Number	Heading
[SWS_CM_10247]	Deserialization of strings
[SWS_CM_10266]	Applicability of mandatory padding after variable length data elements
[SWS_CM_10285]	Responsibility of proper string encoding
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10295]	Providing the received event data
[SWS_CM_10299]	Source of a SOME/IP request message
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10308]	Conditions for sending of a SOME/IP response message
[SWS_CM_10310]	Source of a SOME/IP response message
[SWS_CM_10312]	Content of the SOME/IP response message
[SWS_CM_10313]	Checks for a received SOME/IP response message
[SWS_CM_10317]	Making the Future ready
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10327]	Providing the received event data
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10344]	Content of the SOME/IP response message
[SWS_CM_10358]	Identifying the right application error in a message with Message Type set to `RESPONSE` (0x80)
[SWS_CM_10362]	Raising checked errors for application errors
[SWS_CM_10372]	Inclusion of Implementation Types header files
[SWS_CM_10373]	Implementation Types header files existence
[SWS_CM_10375]	Implementation Types header file namespace
[SWS_CM_10383]	GetHandle function to return the proxy instance creation handle
[SWS_CM_10384]	Change of Service Interface Deployment
[SWS_CM_10385]	Change of Service Instance Deployment
[SWS_CM_10392]	`ScaleLinearAndTexttable` Class Template
[SWS_CM_10393]	`ScaleLinearAndTexttable` static assertion
[SWS_CM_10394]	`ScaleLinearAndTexttable` underlying type deduction
[SWS_CM_10395]	`ScaleLinearAndTexttable` default constructor
[SWS_CM_10396]	`ScaleLinearAndTexttable` copy constructor
[SWS_CM_10397]	`ScaleLinearAndTexttable` constructor with enum class argument
[SWS_CM_10398]	`ScaleLinearAndTexttable` constructor with underlying type argument
[SWS_CM_10399]	`ScaleLinearAndTexttable` copy assignment operator
[SWS_CM_10400]	`ScaleLinearAndTexttable` assignment operator with enum class argument

▽

△

Number	Heading
[SWS_CM_10401]	`ScaleLinearAndTexttable` assignment operator with underlying type argument
[SWS_CM_10402]	`ScaleLinearAndTexttable` cast operator to the underlying type
[SWS_CM_10403]	Equal to operator between two `ScaleLinearAndTexttable` objects
[SWS_CM_10404]	Equal to operators between `ScaleLinearAndTexttable` and an underlying type
[SWS_CM_10405]	Equal to operators between `ScaleLinearAndTexttable` and an enum class
[SWS_CM_10406]	Not equal to operator between two `ScaleLinearAndTexttable` objects
[SWS_CM_10407]	Not equal to operators between `ScaleLinearAndTexttable` and an underlying type
[SWS_CM_10408]	Not equal to operators between `ScaleLinearAndTexttable` and an enum class
[SWS_CM_10409]	Scale Linear And Textable type definition
[SWS_CM_10414]	Initiate a method call
[SWS_CM_10428]	payload representing application error
[SWS_CM_10430]	Handling invalid messages with Message Type set to `RESPONSE` (0x80)
[SWS_CM_10431]	Mapping of ara::core::ErrorCode
[SWS_CM_10432]	
[SWS_CM_10433]	Declaration of Construction Token
[SWS_CM_10434]	Creation of a Construction Token
[SWS_CM_10435]	Exception-less creation of service skeleton using Instance ID
[SWS_CM_10436]	Exception-less creation of service skeleton using Instance Spec
[SWS_CM_10437]	Exception-less creation of service skeleton using Instance ID Container
[SWS_CM_10438]	Exception-less creation of service proxy
[SWS_CM_10450]	`InstanceSpecifier` check during the creation of service skeleton
[SWS_CM_10451]	`InstanceIdentifierContainer` check during the creation of service skeleton
[SWS_CM_10452]	InstanceSpecifier translation to InstanceIdentifiers
[SWS_CM_10590]	Abstract Network Protocol Binding
[SWS_CM_11001]	Mapping of OfferService method
[SWS_CM_11002]	Assigning a DDS DomainParticipant to a Service Instance
[SWS_CM_11006]	Mapping of FindService method
[SWS_CM_11009]	Discovering remote Service Instances through DDS DomainParticipants
[SWS_CM_11015]	Mapping Events to DDS Topics
[SWS_CM_11017]	Mapping of Send method
[SWS_CM_11018]	Mapping of Subscribe method
[SWS_CM_11019]	Creating a DDS DataReader for event subscription
[SWS_CM_11021]	Mapping of Unsubscribe method
[SWS_CM_11023]	Mapping of GetNewSamples method

▽

△

Number	Heading
[SWS_CM_11024]	Mapping of GetFreeSampleCount method
[SWS_CM_11041]	DDS serialization of `StdCppImplementationDataType` of `category` `VALUE`
[SWS_CM_11043]	DDS serialization of `StdCppImplementationDataType` of `category` `STRUCTURE`
[SWS_CM_11044]	DDS serialization of `StdCppImplementationDataType` of `category` `STRING` with string `shortName`
[SWS_CM_11046]	Encoding Format and Endianness of Strings in DDS
[SWS_CM_11047]	DDS serialization of `CppImplementationDataType` of `category` `VECTOR`
[SWS_CM_11048]	DDS serialization of `CppImplementationDataType` of `category` `ARRAY`
[SWS_CM_11102]	DDS Service Reply Topic data type definition
[SWS_CM_11132]	Mapping of Update method
[SWS_CM_11133]	Mapping of Subscribe method
[SWS_CM_11134]	Creating a DDS DataReader for field subscription
[SWS_CM_11136]	Mapping of Unsubscribe method
[SWS_CM_11138]	Mapping of GetNewSamples method
[SWS_CM_11139]	Mapping of GetFreeSampleCount method
[SWS_CM_11145]	DDS Service Request Topic data type definition for Field getter and setter operations
[SWS_CM_11146]	DDS Service Reply Topic data type definition for Field getter and setter operations
[SWS_CM_11264]	Definition general ara::com errors
[SWS_CM_11265]	Use of general ara::com errors
[SWS_CM_11266]	Definition of Application Errors
[SWS_CM_90001]	Restrictions on executing methods
[SWS_CM_90002]	Restrictions on sending events
[SWS_CM_90003]	Restrictions on receiving events
[SWS_CM_90005]	Restrictions on offering services
[SWS_CM_90006]	Restrictions on using services
[SWS_CM_90113]	Behavior of a ServiceSkeleton over TLS before successful completion of the handshake
[SWS_CM_90114]	Behavior of a ServiceSkeleton over DTLS before successful completion of the handshake
[SWS_CM_90118]	Transport of Service communication over an IPsec security association
[SWS_CM_90401]	
[SWS_CM_90402]	
[SWS_CM_90403]	
[SWS_CM_90404]	
[SWS_CM_90405]	
[SWS_CM_90406]	
[SWS_CM_90407]	

▽

△

Number	Heading
[SWS_CM_90408]	
[SWS_CM_90410]	
[SWS_CM_90411]	
[SWS_CM_90412]	
[SWS_CM_90413]	
[SWS_CM_90415]	
[SWS_CM_90416]	
[SWS_CM_90417]	
[SWS_CM_90420]	E2E ProfileCheckStatus of a sample
[SWS_CM_90421]	ara::com::e2e::ProfileCheckStatus
[SWS_CM_90422]	ara::com:E2E_state_machine::E2EState
[SWS_CM_90424]	Provide E2E Result
[SWS_CM_90430]	
[SWS_CM_90431]	
[SWS_CM_90433]	
[SWS_CM_90434]	Provision of a `Fire and Forget method`
[SWS_CM_90435]	Initiate a `Fire and Forget method` call
[SWS_CM_90436]	No checked errors for `Fire and Forget method` calls
[SWS_CM_90437]	Send event where Communication Management is responsible for the data
[SWS_CM_90438]	Allocating data for event transfer
[SWS_CM_90443]	
[SWS_CM_90444]	
[SWS_CM_90445]	
[SWS_CM_90446]	
[SWS_CM_90451]	
[SWS_CM_90452]	

Table E.21: Changed Specification Items in R19-11

E.6.3 Deleted Specification Items in R19-11

Number	Heading
[SWS_CM_00172]	Method to update the event cache
[SWS_CM_00173]	Method to get the cached samples
[SWS_CM_00174]	Method to clean-up the event cache
[SWS_CM_00266]	FilterFunction for incoming event filtering
[SWS_CM_00300]	Event Cache Update Policy

▽

△

Number	Heading
[SWS_CM_00307]	Sample Container
[SWS_CM_10305]	Store the received method data
[SWS_CM_10337]	Store the received method data
[SWS_CM_90409]	
[SWS_CM_90414]	
[SWS_CM_90418]	
[SWS_CM_90419]	
[SWS_CM_90423]	E2EResult
[SWS_CM_90439]	
[SWS_CM_90440]	
[SWS_CM_90441]	
[SWS_CM_90442]	
[SWS_CM_90447]	
[SWS_CM_90448]	
[SWS_CM_90449]	
[SWS_CM_90450]	
[SWS_CM_90453]	
[SWS_CM_90454]	
[SWS_CM_90455]	
[SWS_CM_90456]	
[SWS_CM_90457]	
[SWS_CM_90458]	
[SWS_CM_90459]	
[SWS_CM_90460]	
[SWS_CM_90461]	
[SWS_CM_90462]	
[SWS_CM_90463]	
[SWS_CM_90464]	
[SWS_CM_90465]	
[SWS_CM_90466]	

Table E.22: Deleted Specification Items in R19-11

E.7 Constraint and Specification Item Changes between AUTOSAR Release R18-10 and R19-03

E.7.1 Added Specification Items in 19-03

none

E.7.2 Changed Specification Items in 19-03

none

E.7.3 Deleted Specification Items in 19-03

none

E.8 Constraint and Specification Item Changes between AUTOSAR Release R18-03 and R18-10

E.8.1 Added Specification Items in 18-10

Number	Heading
[SWS_CM_00118]	Method Instance Specifier Translation
[SWS_CM_00134]	Copy semantics of service skeleton class
[SWS_CM_00135]	Move semantics of service skeleton class
[SWS_CM_00136]	Copy semantics of service proxy class
[SWS_CM_00137]	Move semantics of service proxy class
[SWS_CM_00152]	Creation of service skeleton using Instance Spec
[SWS_CM_00153]	Creation of service skeleton using Instance ID Container
[SWS_CM_00317]	Copy semantics of handle Type Class
[SWS_CM_00318]	Move semantics of handle Type Class
[SWS_CM_00333]	Set Subscription State change handler
[SWS_CM_00334]	Unset Subscription State change handler
[SWS_CM_00350]	Instance Specifier Class
[SWS_CM_00452]	Usage of attribute `arraySize` of an `CppImplementationDataType` with category `VECTOR`
[SWS_CM_00502]	`CustomCppImplementationDataType` of `category` `ARRAY`
[SWS_CM_00503]	`StdCppImplementationDataType` of `category` VECTOR with one dimension defined with an `Allocator`
[SWS_CM_00504]	Supported `Primitive Cpp Implementation Data Type`s
[SWS_CM_00505]	`StdCppImplementationDataType` with `category` ASSOCIATIVE_MAP defined with an `Allocator`
[SWS_CM_00506]	`CustomCppImplementationDataType` of category `ASSOCIATIVE_MAP`
[SWS_CM_00507]	`CustomCppImplementationDataType` of category `VECTOR`
[SWS_CM_00508]	`CustomCppImplementationDataType` of `category` `VARIANT`
[SWS_CM_00509]	`StdCppImplementationDataType` with the category `STRING` with a defined `Allocator`
[SWS_CM_00622]	Find service with immediately returned request using Instance Specifier

△

Number	Heading
[SWS_CM_00623]	Find service with handler registration using Instance Specifier
[SWS_CM_01059]	`Variant` destructor
[SWS_CM_01060]	`Variant` move assignment operator
[SWS_CM_01061]	`Variant` default copy assignment operator
[SWS_CM_01062]	`Variant` converting assignment operator
[SWS_CM_01063]	`Variant` function to return the zero-based index of the alternative
[SWS_CM_01064]	`Variant` function to check if the Variant is in invalid state
[SWS_CM_01065]	`Variant` function to swap two Variants
[SWS_CM_01066]	`Variant` function to create a new value in-place, in an existing Variant object
[SWS_CM_01067]	`Variant` function to create a new value in-place, in an existing Variant object using an initializer list
[SWS_CM_01068]	`Variant` function to create a new value in-place, in an existing Variant object by destoying and initializing the contained value
[SWS_CM_01069]	`Variant` function to create a new value in-place, in an existing Variant object by destoying and initializing the contained value using an initializer list
[SWS_CM_10088]	
[SWS_CM_10098]	
[SWS_CM_10099]	
[SWS_CM_10174]	Mix of signal-based and SOME/IP communication
[SWS_CM_10226]	
[SWS_CM_10227]	
[SWS_CM_10250]	
[SWS_CM_10251]	
[SWS_CM_10254]	
[SWS_CM_10255]	
[SWS_CM_10383]	GetHandle function to return the proxy instance creation handle
[SWS_CM_10391]	
[SWS_CM_10392]	`ScaleLinearAndTexttable` Class Template
[SWS_CM_10393]	`ScaleLinearAndTexttable` static assertion
[SWS_CM_10394]	`ScaleLinearAndTexttable` underlying type deduction
[SWS_CM_10395]	`ScaleLinearAndTexttable` default constructor
[SWS_CM_10396]	`ScaleLinearAndTexttable` copy constructor
[SWS_CM_10397]	`ScaleLinearAndTexttable` constructor with enum class argument
[SWS_CM_10398]	`ScaleLinearAndTexttable` constructor with underlying type argument
[SWS_CM_10399]	`ScaleLinearAndTexttable` copy assignment operator
[SWS_CM_10400]	`ScaleLinearAndTexttable` assignment operator with enum class argument
[SWS_CM_10401]	`ScaleLinearAndTexttable` assignment operator with underlying type argument
[SWS_CM_10402]	`ScaleLinearAndTexttable` cast operator to the underlying type

▽

△

Number	Heading
[SWS_CM_10403]	Equal to operator between two`ScaleLinearAndTexttable` objects
[SWS_CM_10404]	Equal to operators between`ScaleLinearAndTexttable` and an underlying type
[SWS_CM_10405]	Equal to operators between `ScaleLinearAndTexttable` and an enum class
[SWS_CM_10406]	Not equal to operator between two`ScaleLinearAndTexttable` objects
[SWS_CM_10407]	Not equal to operators between`ScaleLinearAndTexttable` and an underlying type
[SWS_CM_10408]	Not equal to operators between `ScaleLinearAndTexttable` and an enum class
[SWS_CM_10409]	Scale Linear And Textable type definition
[SWS_CM_10410]	`InstanceIdentifier` check during the creation of service skeleton
[SWS_CM_10411]	Service method processing modes
[SWS_CM_10412]	Invoking GetHandlers
[SWS_CM_10413]	Invoking SetHandlers
[SWS_CM_10414]	Initiate a method call
[SWS_CM_10415]	Notify the Field value after a call to the SetHandler function
[SWS_CM_10428]	payload representing application error
[SWS_CM_10429]	Identifying the right application error in a message with Message Type set to `ERROR` (0x81)
[SWS_CM_10430]	Handling invalid messages with Message Type set to `RESPONSE` (0x81)
[SWS_CM_10431]	Mapping of ara::core::ErrorCode
[SWS_CM_10432]	
[SWS_CM_10433]	Declaration of Construction Token
[SWS_CM_10434]	Creation of a Construction Token
[SWS_CM_10435]	Exception-less creation of service skeleton using Instance ID
[SWS_CM_10436]	Exception-less creation of service skeleton using Instance Spec
[SWS_CM_10437]	Exception-less creation of service skeleton using Instance ID Container
[SWS_CM_10438]	Exception-less creation of service proxy
[SWS_CM_10450]	`InstanceSpecifier` check during the creation of service skeleton
[SWS_CM_10451]	`InstanceIdentifierContainer` check during the creation of service skeleton
[SWS_CM_10452]	InstanceSpecifier translation to InstanceIdentifiers
[SWS_CM_10590]	Abstract Network Protocol Binding
[SWS_CM_11029]	Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Methods in the ServiceInterface
[SWS_CM_11030]	Assigning a DDS Topic and a DDS DataWriter to every Field in the ServiceInterface with its hasNotifier attribute equal to true
[SWS_CM_11031]	Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Field Getters/Setters in the ServiceInterface
[SWS_CM_11040]	DDS standard serialization rules

▽

△

Number	Heading
[SWS_CM_11049]	DDS serialization of CppImplementationDataType of category ASSOCIATIVE_MAP
[SWS_CM_11050]	DDS serialization of CppImplementationDataType of category VARIANT
[SWS_CM_11100]	Mapping Methods to DDS Service Methods and Topics
[SWS_CM_11101]	DDS Service Request Topic data type definition
[SWS_CM_11102]	DDS Service Reply Topic data type definition
[SWS_CM_11103]	Creating a DataWriter to handle method requests on the client side
[SWS_CM_11104]	Creating a DataReader to handle method responses on the client side
[SWS_CM_11105]	Creating a DataReader to handle method requests on the server side
[SWS_CM_11106]	Creating a DataWriter to handle method responses on the server side
[SWS_CM_11107]	Calling a service method from the client side
[SWS_CM_11108]	Notifying the client of a response to a method call
[SWS_CM_11109]	Processing a method call on the server side (event driven)
[SWS_CM_11110]	Creating a DataReaderListener to process asynchronous requests on the server side
[SWS_CM_11111]	Processing a method call on the server side (polling)
[SWS_CM_11112]	Sending a method call response from the server side
[SWS_CM_11130]	Mapping Fields with hasNotifier attribute to DDS Topics
[SWS_CM_11131]	Field Notifier DDS Topic data type definition
[SWS_CM_11132]	Mapping of Send method
[SWS_CM_11133]	Mapping of Subscribe method
[SWS_CM_11134]	Creating a DDS DataReader for field subscription
[SWS_CM_11135]	Creating a DDS DataReaderListener for field subscription
[SWS_CM_11136]	Mapping of Unsubscribe method
[SWS_CM_11137]	Mapping of GetSubscriptionState method
[SWS_CM_11138]	Mapping of Update method
[SWS_CM_11139]	Mapping of GetCachedSamples method
[SWS_CM_11140]	Mapping of SetReceiveHandler method
[SWS_CM_11141]	Mapping of UnsetReceiveHandler method
[SWS_CM_11142]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_11143]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_11144]	Mapping of Field Get/Set methods to DDS Service Methods and Topics
[SWS_CM_11145]	DDS Service Request Topic data type definition for Field getter and setter operations
[SWS_CM_11146]	DDS Service Reply Topic data type definition for Field getter and setter operations
[SWS_CM_11147]	Creating a DataWriter to handle get/set requests on the client side
[SWS_CM_11148]	Creating a DataReader to handle get/set responses on the client side
[SWS_CM_11149]	Creating a DataReader to handle get/set requests on the server side

▽

△

Number	Heading
[SWS_CM_11150]	Creating a DataWriter to handle get/set responses on the server side
[SWS_CM_11151]	Calling get/set method associated with a field from the client side
[SWS_CM_11152]	Notifying the client of the response to the get/set method call
[SWS_CM_11153]	Processing a get/set method call associated with a field on the server side (event driven)
[SWS_CM_11154]	Creating a DataReaderListener to process asynchronous requests for field getters and setters on the server side
[SWS_CM_11155]	Processing a get/set method call associated with a field on the server side (polling)
[SWS_CM_11156]	Sending a response for a get/set method call associated with a field from the server side
[SWS_CM_11264]	Definition general ara::com errors
[SWS_CM_11265]	Use of general ara::com errors
[SWS_CM_11266]	Definition of Application Errors
[SWS_CM_90005]	Restrictions on offering services
[SWS_CM_90006]	Restrictions on using services
[SWS_CM_90111]	Behavior of a ServiceProxy over TLS before successful completion of the handshake
[SWS_CM_90112]	Behavior of a ServiceProxy over DTLS before successful completion of the handshake
[SWS_CM_90113]	Behavior of a ServiceSkeleton over TLS before successful completion of the handshake
[SWS_CM_90114]	Behavior of a ServiceSkeleton over DTLS before successful completion of the handshake
[SWS_CM_90115]	SecOC secure channel for methods using unreliable transport
[SWS_CM_90116]	SecOC secure channel for events using unreliable transport
[SWS_CM_90117]	IPsec secure channel between communication nodes
[SWS_CM_90118]	Transport of Service communication over an IPsec security association
[SWS_CM_90119]	Behavior of a creating ServiceProxy over TLS or DTLS
[SWS_CM_90120]	TLS client role of a Proxy
[SWS_CM_90121]	TLS server role of a Skeleton
[SWS_CM_90201]	Secure channel creation
[SWS_CM_90202]	Using secure channels
[SWS_CM_90203]	TLS secure channel for methods using reliable transport
[SWS_CM_90204]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90205]	TLS secure channel for events using reliable transport
[SWS_CM_90206]	DTLS secure channel for events using unreliable transport
[SWS_CM_90207]	TLS secure channel for fields
[SWS_CM_90209]	IPsec secure channel between communication nodes and Transport of Service communication over an IPsec security association

▽

△

Number	Heading
[SWS_CM_90210]	Using the DDS Security standard plug-ins in the Adaptive Platform

Table E.23: Added Specification Items in 18-10

E.8.2 Changed Specification Items in 18-10

Number	Heading
[SWS_CM_00102]	Uniqueness of offered service
[SWS_CM_00103]	Protocol where a service is offered
[SWS_CM_00112]	Method to get the value of a field
[SWS_CM_00113]	Method to set the value of a field
[SWS_CM_00114]	Registering Getters
[SWS_CM_00116]	Registering Setters
[SWS_CM_00120]	Provision of an update notification event for a Field
[SWS_CM_00122]	Find service with immediately returned request using Instance ID
[SWS_CM_00123]	Find service with handler registration using Instance ID
[SWS_CM_00124]	Find service handler behavior
[SWS_CM_00128]	Ensuring the existence of valid Field values
[SWS_CM_00129]	Ensuring the existence of SetHandler
[SWS_CM_00130]	Creation of service skeleton using Instance ID
[SWS_CM_00131]	Creation of service proxy
[SWS_CM_00172]	Method to update the event cache
[SWS_CM_00191]	Provision of method
[SWS_CM_00192]	Synchronous behavior of method call
[SWS_CM_00193]	Asynchronous behavior of method call with polling
[SWS_CM_00194]	Cancel the method call
[SWS_CM_00195]	Retrieving results of the method call
[SWS_CM_00196]	Initiate a method call
[SWS_CM_00197]	Asynchronous behavior of method call with notification
[SWS_CM_00198]	Set service method processing mode
[SWS_CM_00199]	Process Service method invocation
[SWS_CM_00202]	SOME/IP FindService message
[SWS_CM_00203]	SOME/IP OfferService message
[SWS_CM_00205]	Content of SOME/IP SubscribeEventgroup message
[SWS_CM_00206]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_00207]	Content of SOME/IP StopSubscribeEventgroup message
[SWS_CM_00208]	SOME/IP SubscribeEventgroupNack message

▽

△

Number	Heading
[SWS_CM_00257]	
[SWS_CM_00258]	
[SWS_CM_00264]	
[SWS_CM_00302]	Instance Identifier Class
[SWS_CM_00304]	Service Handle Container
[SWS_CM_00306]	Sample Pointer
[SWS_CM_00307]	Sample Container
[SWS_CM_00312]	Handle Type Class
[SWS_CM_00314]	Call SubscriptionStateChangeHandler with kSubscribed
[SWS_CM_00315]	Re-establishing an active subscription
[SWS_CM_00316]	Query Subscription State
[SWS_CM_00383]	Find Service Handler
[SWS_CM_00400]	Naming of data types by short name
[SWS_CM_00402]	Primitive fixed width integer types
[SWS_CM_00403]	StdCppImplementationDataType of category ARRAY with one dimension
[SWS_CM_00404]	Array Data Type with more than one dimension
[SWS_CM_00405]	Structure Data Type
[SWS_CM_00406]	StdCppImplementationDataType with the category STRING
[SWS_CM_00407]	StdCppImplementationDataType of category VECTOR with one dimension defined without an Allocator
[SWS_CM_00408]	Vector Data Type with more than one dimension
[SWS_CM_00409]	StdCppImplementationDataType with category ASSOCIATIVE_MAP defined without an Allocator
[SWS_CM_00410]	Data Type redefinition
[SWS_CM_00411]	Avoid Data Type redeclaration
[SWS_CM_00414]	Element specification typed by CppImplementationDataType
[SWS_CM_00421]	Provide data type definitions
[SWS_CM_00423]	Data Type Mapping
[SWS_CM_00424]	Enumeration Data Type
[SWS_CM_00425]	Definition of enumerators
[SWS_CM_00426]	Reject incompleteEnumeration Data Types
[SWS_CM_00449]	Variant Data Type
[SWS_CM_00450]	Define the maximum size of allocated vector memory
[SWS_CM_01004]	Inclusion of common header file
[SWS_CM_01008]	Namespace for Service Identifier Type definitions
[SWS_CM_01010]	Service Identifier and Service Version Classes
[SWS_CM_01015]	Service methods namespace
[SWS_CM_01019]	Data Type declarations in Types header file
[SWS_CM_01020]	Folder structure

▽

△

Number	Heading
[SWS_CM_01032]	Accessing optional record elements inside a`Structure Cpp Implementation Data Type` that are serialized with the Tag-Length-Value principle.
[SWS_CM_01045]	Use cases for the definition of`tlvDataId`
[SWS_CM_01046]	Definition of`tlvDataId`
[SWS_CM_01049]	Synchronization of`tlvDataId`s between the interacting proxy and skeleton instances.
[SWS_CM_01050]	`Variant` Class Template
[SWS_CM_01054]	`Variant` converting constructor
[SWS_CM_01055]	`Variant` explicit converting constructor with specified alternative
[SWS_CM_01056]	`Variant` explicit converting constructor with specified alternative and initializer list
[SWS_CM_01057]	`Variant` explicit converting constructor with alternative specified by index
[SWS_CM_01058]	`Variant` explicit converting constructor with alternative specified by index and initializer list
[SWS_CM_10017]	
[SWS_CM_10036]	
[SWS_CM_10042]	
[SWS_CM_10059]	
[SWS_CM_10070]	
[SWS_CM_10234]	
[SWS_CM_10235]	
[SWS_CM_10242]	Model representation of UTF-8 Strings
[SWS_CM_10245]	Serialization of strings
[SWS_CM_10247]	Deserialization of strings
[SWS_CM_10253]	
[SWS_CM_10262]	Insertion of an associative map length field
[SWS_CM_10265]	Serialization of associative map elements
[SWS_CM_10285]	Responsibility of proper string encoding
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10294]	Deserializing the payload
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10304]	Deserializing the payload
[SWS_CM_10308]	Conditions for sending of a SOME/IP response message
[SWS_CM_10312]	Content of the SOME/IP response message
[SWS_CM_10313]	Checks for a received SOME/IP response message
[SWS_CM_10316]	Deserializing the payload - normal response messages
[SWS_CM_10317]	Making the Future ready
[SWS_CM_10323]	Content of the SOME/IP event message

▽

△

Number	Heading
[SWS_CM_10324]	Checks for a received SOME/IP event message
[SWS_CM_10326]	Deserializing the payload
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10336]	Deserializing the payload
[SWS_CM_10339]	Invoke the registered set/get handlers - polling
[SWS_CM_10344]	Content of the SOME/IP response message
[SWS_CM_10345]	Checks for a received SOME/IP response message
[SWS_CM_10348]	Deserializing the payload
[SWS_CM_10349]	Making the Future ready
[SWS_CM_10357]	Distinguishing errors from normal responses
[SWS_CM_10358]	Identifying the right application error in a message with Message Type set to `RESPONSE` (0x80)
[SWS_CM_10361]	
[SWS_CM_10362]	Raising checked errors for application errors
[SWS_CM_10370]	Common header file for Application Errors
[SWS_CM_10371]	Context of return checked errors
[SWS_CM_10372]	Inclusion of Implementation Types header files
[SWS_CM_10373]	Implementation Types header files existence
[SWS_CM_10374]	Data Type definitions for AUTOSAR Data Types in Implementation Types header files
[SWS_CM_10375]	Implementation Types header file namespace
[SWS_CM_10382]	Calling stop find service for already stopped finds
[SWS_CM_10388]	Enabling of data accumulation for UDP data transmission
[SWS_CM_10389]	Configuration of a data accumulation on a `ProvidedServiceInstance` for transmission over UDP
[SWS_CM_10390]	Configuration of a data accumulation on a `RequiredSomeipServiceInstance` for transmission over UDP
[SWS_CM_11001]	Mapping of OfferService method
[SWS_CM_11002]	Assigning a DDS DomainParticipant to a Service Instance
[SWS_CM_11003]	Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface
[SWS_CM_11004]	Adding Service and Service Instance IDs to the DDS DomainParticipant's USER_DATA QoS Policy
[SWS_CM_11005]	Mapping of StopOfferService method
[SWS_CM_11006]	Mapping of FindService method
[SWS_CM_11007]	Finding a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_11009]	Discovering remote Service Instances through DDS DomainParticipants
[SWS_CM_11010]	Mapping of StartFindService method
[SWS_CM_11011]	Defining a DDS BuiltinParticipantListener

▽

△

Number	Heading
[SWS_CM_11012]	Binding a BuiltinParticipantListener to a DDS DomainParticipant
[SWS_CM_11014]	Unbinding a BuiltinParticipantListener from a DDS DomainParticipant
[SWS_CM_11015]	Mapping Events to DDS Topics
[SWS_CM_11016]	DDS Topic data type definition
[SWS_CM_11017]	Mapping of Send method
[SWS_CM_11018]	Mapping of Subscribe method
[SWS_CM_11019]	Creating a DDS DataReader for event subscription
[SWS_CM_11020]	Defining a DDS DataReaderListener
[SWS_CM_11021]	Mapping of Unsubscribe method
[SWS_CM_11022]	Mapping of GetSubscriptionState method
[SWS_CM_11023]	Mapping of Update method
[SWS_CM_11025]	Mapping of SetReceiveHandler method
[SWS_CM_11026]	Mapping of UnsetReceiveHandler method
[SWS_CM_11027]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_11028]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_11041]	DDS serialization of `StdCppImplementationDataType` of `category` `VALUE`
[SWS_CM_11042]	DDS serialization of enumeration data types
[SWS_CM_11043]	DDS serialization of `StdCppImplementationDataType` of `category` `STRUCTURE`
[SWS_CM_11044]	DDS serialization of `StdCppImplementationDataType` of `category` `STRING` with string `shortName`
[SWS_CM_11046]	Encoding Format and Endianness of Strings in DDS
[SWS_CM_11047]	DDS serialization of `CppImplementationDataType` of `category`VECTOR
[SWS_CM_11048]	DDS serialization of `CppImplementationDataType` of `category`ARRAY
[SWS_CM_90001]	Restrictions on executing methods
[SWS_CM_90101]	Secure UDP and TCP channel creation for TLS, DTLS and SecOC
[SWS_CM_90102]	Using secure TLS, DTLS and SecOC channels
[SWS_CM_90103]	TLS secure channel for methods using reliable transport
[SWS_CM_90104]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90105]	TLS secure channel for events using reliable transport
[SWS_CM_90106]	DTLS secure channel for events using unreliable transport
[SWS_CM_90108]	SecOC secure channel for methods using reliable transport
[SWS_CM_90109]	SecOC secure channel for events using reliable transport
[SWS_CM_90110]	SecOC secure channel for fields
[SWS_CM_90401]	
[SWS_CM_90404]	
[SWS_CM_90420]	E2ECheckStatus of a sample
[SWS_CM_90421]	ara::com:E2E_state_machine::E2Echeckstatus

▽

△

Number	Heading
[SWS_CM_90422]	ara::com:E2E_state_machine::E2EState
[SWS_CM_90430]	
[SWS_CM_90436]	No checked errors for `Fire and Forget method` calls

Table E.24: Changed Specification Items in 18-10

E.8.3 Deleted Specification Items in 18-10

Number	Heading
[SWS_CM_00262]	
[SWS_CM_00263]	
[SWS_CM_00305]	Find Service Handler
[SWS_CM_00320]	`FutureStatus`
[SWS_CM_00321]	`Future` Class Template
[SWS_CM_00322]	`Future` default constructor
[SWS_CM_00323]	`Future` move constructor
[SWS_CM_00324]	`Future` unwrapping constructor
[SWS_CM_00325]	Move assignment operator
[SWS_CM_00326]	`Future::get`
[SWS_CM_00327]	`Future::valid`
[SWS_CM_00328]	`Future::wait`
[SWS_CM_00329]	`Future::wait_for`
[SWS_CM_00330]	`Future::wait_until`
[SWS_CM_00331]	`Future::then`
[SWS_CM_00332]	`Future::is_ready`
[SWS_CM_00340]	`Promise` Class Template
[SWS_CM_00341]	`Promise` default constructor
[SWS_CM_00342]	`Promise` move constructor
[SWS_CM_00343]	`Promise` move assignment operator
[SWS_CM_00344]	`Promise::get_future`
[SWS_CM_00345]	`Promise::set_value`
[SWS_CM_00346]	`Promise::set_value`, forwarding reference version
[SWS_CM_00347]	`Promise::set_exception`
[SWS_CM_00348]	`Promise::set_future_dtor_handler`
[SWS_CM_00401]	Naming of data types by symbol
[SWS_CM_00412]	Union Data Type
[SWS_CM_00413]	Element specification typed by Base Type

▽

△

Number	Heading
[SWS_CM_00415]	Element specification typed by Array
[SWS_CM_00416]	Element specification typed by Structure
[SWS_CM_00417]	Element specification typed by Union
[SWS_CM_00418]	Element specification typed by Vector
[SWS_CM_00419]	Element specification typed by Map
[SWS_CM_00420]	Element specification typed by String Data Type with `baseTypeSize` of 8
[SWS_CM_00422]	Reject data type definitions
[SWS_CM_00427]	String Data Type with `baseTypeSize` of 16
[SWS_CM_00428]	Element specification typed by String Data Type with `baseTypeSize` of 16
[SWS_CM_00448]	Element specification typed by Variant
[SWS_CM_00451]	Namespace specification for an ImplementationDataType of category VECTOR
[SWS_CM_01033]	`Optional` Class Template
[SWS_CM_01034]	`Optional` default constructor
[SWS_CM_01035]	`Optional` move constructor
[SWS_CM_01036]	`Optional` copy constructor
[SWS_CM_01037]	`Optional` destructor
[SWS_CM_01038]	`Optional` move assignment operator
[SWS_CM_01039]	`Optional` default copy assignment operator
[SWS_CM_01040]	`Optional` function to get contained value
[SWS_CM_01041]	`Optional` function to check availability of contained value
[SWS_CM_01042]	`Optional` bool operator
[SWS_CM_01043]	`Optional` reset function
[SWS_CM_01044]	
[SWS_CM_10040]	
[SWS_CM_10243]	UTF-16BE Strings
[SWS_CM_10244]	UTF-16LE Strings
[SWS_CM_10286]	Encoding mismatch in input configurations
[SWS_CM_10351]	Service application errors
[SWS_CM_10352]	Definition of `ServiceNotAvailableException`
[SWS_CM_10353]	Use of `ServiceNotAvailableException`
[SWS_CM_10354]	Definition of `ApplicationErrorException`
[SWS_CM_10355]	Use of `ApplicationErrorException`
[SWS_CM_10356]	Definition of sub-classes of `ApplicationErrorException`
[SWS_CM_10359]	Deserializing the payload - error response mesages
[SWS_CM_11045]	Serialization of Strings of `baseTypeSize` 16
[SWS_CM_90432]	Functionality of Sample Pointer

Table E.25: Deleted Specification Items in 18-10

E.9 Constraint and Specification Item Changes between AUTOSAR Release R17-10 and R18-03

E.9.1 Added Specification Items in 18-03

Number	Heading
[SWS_CM_00008]	Service proxy Field class
[SWS_CM_00172]	Method to update the event cache
[SWS_CM_00173]	Method to get the cached samples
[SWS_CM_00174]	Method to clean-up the event cache
[SWS_CM_00313]	Call SubscriptionStateChangeHandler with kSubscriptionPending
[SWS_CM_00314]	Call SubscriptionStateChangeHandler with kSubscribed
[SWS_CM_00315]	Re-establishing an active subscription
[SWS_CM_00316]	Query Subscription State
[SWS_CM_00383]	Extended Find Service Handler
[SWS_CM_00412]	Union Data Type
[SWS_CM_00417]	Element specification typed by Union
[SWS_CM_00448]	Element specification typed by Variant
[SWS_CM_00449]	Variant Data Type
[SWS_CM_00450]	Maximum size of allocated vector memory
[SWS_CM_00451]	Namespace specification for an ImplementationDataType of category VECTOR
[SWS_CM_01032]	Accessing optional record elements inside a `Structure Cpp Implementation Data Type` that are serialized with the Tag-Length-Value principle.
[SWS_CM_01033]	`Optional` Class Template
[SWS_CM_01034]	`Optional` default constructor
[SWS_CM_01035]	`Optional` move constructor
[SWS_CM_01036]	`Optional` copy constructor
[SWS_CM_01037]	`Optional` destructor
[SWS_CM_01038]	`Optional` move assignment operator
[SWS_CM_01039]	`Optional` default copy assignment operator
[SWS_CM_01040]	`Optional` function to get contained value
[SWS_CM_01041]	`Optional` function to check availability of contained value
[SWS_CM_01042]	`Optional` bool operator
[SWS_CM_01043]	`Optional` reset function
[SWS_CM_01044]	
[SWS_CM_01045]	Every record element inside a struct that contains at least one optional record element shall be serialized based on the Tag-Length-Value principle.

▽

△

Number	Heading
[SWS_CM_01046]	Regarding the definition of `tlvDataId` see [TPS_MANI_01097] and [constr_1532] for details.
[SWS_CM_01047]	Every record element shall have a `wire type` assigned when the optionality is used for at least one record element inside the struct.
[SWS_CM_01048]	Every record element shall have a `tag` assigned when the optionality is used for at least one record element inside the struct.
[SWS_CM_01049]	The `tlvDataId`s shall be synchronized between the interacting proxy and skeleton instances.
[SWS_CM_01050]	`Variant` Class Template
[SWS_CM_01051]	`Variant` default constructor
[SWS_CM_01052]	`Variant` move constructor
[SWS_CM_01053]	`Variant` copy constructor
[SWS_CM_01054]	`Variant` destructor
[SWS_CM_01055]	`Variant` move assignment operator
[SWS_CM_01056]	`Variant` default copy assignment operator
[SWS_CM_01057]	`Variant` function to return the zero-based index of the alternative
[SWS_CM_01058]	`Variant` function to check if the Variant is in invalid state
[SWS_CM_10040]	
[SWS_CM_10235]	
[SWS_CM_10244]	UTF-16LE Strings
[SWS_CM_10372]	Inclusion of Implementation Types header files
[SWS_CM_10373]	Implementation Types header files existence
[SWS_CM_10374]	Data Type definitions for AUTOSAR Data Types in Implementation Types header files
[SWS_CM_10375]	Implementation Types header file namespace
[SWS_CM_10376]	Skip `CompuScale`s with non-point range
[SWS_CM_10377]	Sending SOME/IP SubscribeEventgroup messages - initial
[SWS_CM_10378]	Sending SOME/IP StopSubscribeEventgroup messages
[SWS_CM_10379]	Silently discarding SOME/IP event messages for unsubscribed events
[SWS_CM_10380]	Silently discarding SOME/IP event messages for unsubscribed events
[SWS_CM_10381]	Sending SOME/IP SubscribeEventgroup messages - renewal
[SWS_CM_10382]	Calling stop find service for already stopped finds
[SWS_CM_10384]	Change of Service Interface Deployment
[SWS_CM_10385]	Change of Service Instance Deployment
[SWS_CM_10386]	Change of Network Configuration
[SWS_CM_10387]	Data accumulation for UDP data transmission
[SWS_CM_10388]	Enabling of data accumulation for UDP data transmission
[SWS_CM_10389]	Configuration of a data accumulation on a `ProvidedServiceInstance` for transmission over UDP
[SWS_CM_10390]	Configuration of a data accumulation on a `RequiredSomeipServiceInstance` for transmission over UDP

▽

△

Number	Heading
[SWS_CM_11000]	
[SWS_CM_11001]	Mapping of OfferService method
[SWS_CM_11002]	Assigning a DDS DomainParticipant to a Service Instance
[SWS_CM_11003]	Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface
[SWS_CM_11004]	Adding Service and Service Instance IDs to the DDS Domain Participant's USER_DATA QoS Policy
[SWS_CM_11005]	Mapping of StopOfferService method
[SWS_CM_11006]	Mapping of FindService method
[SWS_CM_11007]	Finding a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_11008]	Creating a DDS DomainParticipant suitable for performing client-side operations
[SWS_CM_11009]	Discovering remote Service Instances through DDS DomainParticipants
[SWS_CM_11010]	Mapping of StartFindService method
[SWS_CM_11011]	Defining a DDS BuiltinParticipantListener
[SWS_CM_11012]	Binding a BuiltinParticipantListener to a DDS DomainParticipant
[SWS_CM_11013]	Mapping of StopFindService method
[SWS_CM_11014]	Unbinding a BuiltinParticipantListener from a DDS DomainParticipant
[SWS_CM_11015]	Mapping Events to DDS Topics
[SWS_CM_11016]	DDS Topic datatype definition
[SWS_CM_11017]	Mapping of Send method
[SWS_CM_11018]	Mapping of Subscribe method
[SWS_CM_11019]	Creating a DDS DataReader for event subscription
[SWS_CM_11020]	Defining a DDS DataReaderListener
[SWS_CM_11021]	Mapping of Unsubscribe method
[SWS_CM_11022]	Mapping of GetSubscriptionState method
[SWS_CM_11023]	Mapping of Update method
[SWS_CM_11024]	Mapping of GetCachedSamples method
[SWS_CM_11025]	Mapping of SetReceiveHandler method
[SWS_CM_11026]	Mapping of UnsetReceiveHandler method
[SWS_CM_11027]	Mapping of SetSubscriptionStateHandler method
[SWS_CM_11028]	Mapping of UnsetSubscriptionStateHandler method
[SWS_CM_11041]	
[SWS_CM_11042]	
[SWS_CM_11043]	
[SWS_CM_11044]	Serialization of Strings of `baseTypeSize` 8
[SWS_CM_11045]	Serialization of Strings of `baseTypeSize` 16
[SWS_CM_11046]	Serialization of `ImplementationDataType` of `category` `VECTOR`
[SWS_CM_11047]	Serialization of `ImplementationDataType` of `category` `ARRAY`

▽

△

Number	Heading
[SWS_CM_11048]	
[SWS_CM_90001]	Restrictions on executing methods
[SWS_CM_90002]	Restrictions on sending events
[SWS_CM_90003]	Restrictions on receiving events
[SWS_CM_90004]	Process separation of network and language binding for access control
[SWS_CM_90433]	
[SWS_CM_90434]	Provision of a `Fire and Forget method`
[SWS_CM_90435]	Initiate a `Fire and Forget method` call
[SWS_CM_90436]	No checked exceptions thrown for `Fire and Forget method` calls
[SWS_CM_90437]	Send event where Communication Management is responsible for the data
[SWS_CM_90438]	Allocating data for event transfer

Table E.26: Added Specification Items in 18-03

E.9.2 Changed Specification Items in 18-03

Number	Heading
[SWS_CM_00002]	Service skeleton class
[SWS_CM_00003]	Service skeleton Event class
[SWS_CM_00004]	Service proxy class
[SWS_CM_00005]	Service proxy Event class
[SWS_CM_00006]	Service proxy Method class
[SWS_CM_00007]	Service skeleton Field class
[SWS_CM_00102]	Uniqueness of offered service
[SWS_CM_00120]	Provision of an update notification event for a Field
[SWS_CM_00123]	Find service with handler registration
[SWS_CM_00124]	Find service handler behavior
[SWS_CM_00141]	Method to subscribe to a service event
[SWS_CM_00162]	Send event where application is responsible for the data
[SWS_CM_00201]	Start of service discovery protocol on Server side
[SWS_CM_00202]	SOME/IP FindService message
[SWS_CM_00203]	SOME/IP OfferService message
[SWS_CM_00204]	SOME/IP StopOffer message
[SWS_CM_00205]	Content of SOME/IP SubscribeEventgroup message
[SWS_CM_00206]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_00207]	Content of SOME/IP StopSubscribeEventgroup message
[SWS_CM_00208]	SOME/IP SubscribeEventgroupNack message

▽

△

Number	Heading
[SWS_CM_00209]	Start of service discovery protocol on Client side
[SWS_CM_00252]	
[SWS_CM_00253]	
[SWS_CM_00254]	
[SWS_CM_00255]	
[SWS_CM_00256]	
[SWS_CM_00257]	
[SWS_CM_00258]	
[SWS_CM_00259]	
[SWS_CM_00260]	
[SWS_CM_00262]	
[SWS_CM_00263]	
[SWS_CM_00264]	
[SWS_CM_00265]	
[SWS_CM_00302]	Instance Identifier Class
[SWS_CM_00303]	Find Service Handle
[SWS_CM_00306]	Sample Pointer
[SWS_CM_00310]	Subscription State
[SWS_CM_00311]	Subscription State Changed Handler
[SWS_CM_00312]	Handle Type Class
[SWS_CM_00400]	Naming of data types by short name
[SWS_CM_00401]	Naming of data types by symbol
[SWS_CM_00402]	Primitive Data Type
[SWS_CM_00403]	Array Data Type with one dimension
[SWS_CM_00404]	Array Data Type with more than one dimension
[SWS_CM_00405]	Structure Data Type
[SWS_CM_00406]	String Data Type with `baseTypeSize` of 8
[SWS_CM_00407]	Vector Data Type with one dimension
[SWS_CM_00408]	Vector Data Type with more than one dimension
[SWS_CM_00409]	Associative Map Data Type
[SWS_CM_00410]	Data Type redefinition
[SWS_CM_00411]	Avoid Data Type redeclaration
[SWS_CM_00413]	Element specification typed by Base Type
[SWS_CM_00414]	Element specification typed by Implementation Data Type
[SWS_CM_00415]	Element specification typed by Array
[SWS_CM_00416]	Element specification typed by Structure
[SWS_CM_00418]	Element specification typed by Vector
[SWS_CM_00419]	Element specification typed by Map
[SWS_CM_00420]	Element specification typed by String Data Type with `baseTypeSize` of 8

▽

△

Number	Heading
[SWS_CM_00421]	Provide data type definitions
[SWS_CM_00422]	Reject data type definitions
[SWS_CM_00423]	Data Type Mapping
[SWS_CM_00424]	Enumeration Data Type
[SWS_CM_00425]	Definition of enumerators
[SWS_CM_00426]	Reject incomplete `Enumeration Data Type`s
[SWS_CM_00427]	String Data Type with `baseTypeSize` of 16
[SWS_CM_00428]	Element specification typed by String Data Type with `baseTypeSize` of 16
[SWS_CM_01005]	Namespace of Service header files
[SWS_CM_01008]	Common header file namespace
[SWS_CM_01010]	Service Identifier and Service Version Classes
[SWS_CM_01015]	Service methods namespace
[SWS_CM_01017]	Service Identifier Type definitions in Common header file
[SWS_CM_01020]	Folder structure
[SWS_CM_01031]	Service fields namespace
[SWS_CM_10013]	
[SWS_CM_10016]	
[SWS_CM_10017]	
[SWS_CM_10034]	
[SWS_CM_10036]	
[SWS_CM_10037]	
[SWS_CM_10042]	
[SWS_CM_10053]	
[SWS_CM_10054]	
[SWS_CM_10055]	
[SWS_CM_10056]	
[SWS_CM_10057]	
[SWS_CM_10058]	
[SWS_CM_10059]	
[SWS_CM_10060]	
[SWS_CM_10070]	
[SWS_CM_10072]	
[SWS_CM_10076]	
[SWS_CM_10169]	
[SWS_CM_10172]	
[SWS_CM_10218]	
[SWS_CM_10219]	
[SWS_CM_10222]	
[SWS_CM_10234]	

▽

△

Number	Heading
[SWS_CM_10242]	UTF-8 Strings
[SWS_CM_10243]	UTF-16BE Strings
[SWS_CM_10245]	Serialization of strings
[SWS_CM_10247]	Deserialization of strings
[SWS_CM_10248]	
[SWS_CM_10252]	
[SWS_CM_10253]	
[SWS_CM_10256]	
[SWS_CM_10257]	
[SWS_CM_10258]	
[SWS_CM_10259]	
[SWS_CM_10260]	
[SWS_CM_10261]	Serialization of an associative map
[SWS_CM_10262]	Insertion of an associative map length field
[SWS_CM_10264]	Size of the associative map length field
[SWS_CM_10265]	Serialization of associative map elements
[SWS_CM_10266]	Applicability of mandatory padding after variable length data elements
[SWS_CM_10267]	Insertion of an associative map length field
[SWS_CM_10268]	
[SWS_CM_10269]	
[SWS_CM_10270]	
[SWS_CM_10271]	
[SWS_CM_10272]	
[SWS_CM_10273]	
[SWS_CM_10274]	
[SWS_CM_10275]	
[SWS_CM_10276]	
[SWS_CM_10277]	
[SWS_CM_10278]	
[SWS_CM_10279]	
[SWS_CM_10280]	
[SWS_CM_10281]	
[SWS_CM_10282]	
[SWS_CM_10283]	
[SWS_CM_10284]	
[SWS_CM_10285]	Responsibility of proper string encoding
[SWS_CM_10286]	Encoding mismatch in input configurations
[SWS_CM_10287]	Conditions for sending of a SOME/IP event message
[SWS_CM_10288]	Transport protocol for sending of a SOME/IP event message

▽

△

Number	Heading
[SWS_CM_10289]	Source of a SOME/IP event message
[SWS_CM_10290]	Destination of a SOME/IP event message
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10293]	Identifying the right event
[SWS_CM_10294]	Deserializing the payload
[SWS_CM_10295]	Store the received event data
[SWS_CM_10296]	Invoke receive handler
[SWS_CM_10297]	Conditions for sending of a SOME/IP request message
[SWS_CM_10298]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_10299]	Source of a SOME/IP request message
[SWS_CM_10300]	Destination of a SOME/IP request message
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10303]	Identifying the right method
[SWS_CM_10304]	Deserializing the payload
[SWS_CM_10305]	Store the received method data
[SWS_CM_10306]	Invoke the method - event driven
[SWS_CM_10307]	Invoke the method - polling
[SWS_CM_10308]	Conditions for sending of a SOME/IP response message
[SWS_CM_10309]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_10310]	Source of a SOME/IP response message
[SWS_CM_10311]	Destination of a SOME/IP response message
[SWS_CM_10312]	Content of the SOME/IP response message
[SWS_CM_10313]	Checks for a received SOME/IP response message
[SWS_CM_10314]	Identifying the right method
[SWS_CM_10315]	Discarding orphaned responses
[SWS_CM_10316]	Deserializing the payload - response messages
[SWS_CM_10317]	Making the Future ready
[SWS_CM_10318]	Invoke the notification function
[SWS_CM_10319]	Conditions for sending of a SOME/IP event message
[SWS_CM_10320]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10321]	Source of a SOME/IP event message
[SWS_CM_10322]	Destination of a SOME/IP event message
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10324]	Checks for a received SOME/IP event message
[SWS_CM_10325]	Identifying the right event
[SWS_CM_10326]	Deserializing the payload

▽

△

Number	Heading
[SWS_CM_10327]	Store the received event data
[SWS_CM_10328]	Invoke receive handler
[SWS_CM_10329]	Conditions for sending of a SOME/IP request message
[SWS_CM_10330]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_10331]	Source of a SOME/IP request message
[SWS_CM_10332]	Destination of a SOME/IP request message
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10335]	Identifying the right method
[SWS_CM_10336]	Deserializing the payload
[SWS_CM_10337]	Store the received method data
[SWS_CM_10338]	Invoke the registered set/get handlers - event driven
[SWS_CM_10339]	Invoke the registered set/get handlers - polling
[SWS_CM_10340]	Conditions for sending of a SOME/IP response message
[SWS_CM_10341]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_10342]	Source of a SOME/IP response message
[SWS_CM_10343]	Destination of a SOME/IP response message
[SWS_CM_10344]	Content of the SOME/IP response message
[SWS_CM_10345]	Checks for a received SOME/IP response message
[SWS_CM_10346]	Identifying the right method
[SWS_CM_10347]	Discarding orphaned responses
[SWS_CM_10348]	Deserializing the payload
[SWS_CM_10349]	Making the Future ready
[SWS_CM_10350]	Invoke the notification function
[SWS_CM_10356]	Definition of sub-classes of `ApplicationErrorException`
[SWS_CM_10357]	Distinguishing errors from normal responses
[SWS_CM_10358]	Identifying the right application error
[SWS_CM_10359]	Deserializing the payload - error response mesages
[SWS_CM_10361]	
[SWS_CM_11262]	
[SWS_CM_11263]	
[SWS_CM_90103]	TLS secure channel for methods using reliable transport
[SWS_CM_90104]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90105]	TLS secure channel for events using reliable transport
[SWS_CM_90106]	DTLS secure channel for events using unreliable transport
[SWS_CM_90401]	
[SWS_CM_90402]	
[SWS_CM_90403]	

▽

△

Number	Heading
[SWS_CM_90404]	
[SWS_CM_90405]	
[SWS_CM_90406]	
[SWS_CM_90407]	
[SWS_CM_90408]	
[SWS_CM_90409]	
[SWS_CM_90410]	
[SWS_CM_90411]	
[SWS_CM_90412]	
[SWS_CM_90413]	
[SWS_CM_90414]	
[SWS_CM_90416]	
[SWS_CM_90417]	
[SWS_CM_90418]	
[SWS_CM_90419]	
[SWS_CM_90420]	E2ECheckStatus of a sample
[SWS_CM_90421]	ara::com:E2E_state_machine::E2Echeckstatus
[SWS_CM_90422]	ara::com:E2E_state_machine::E2EState
[SWS_CM_90423]	E2EResult
[SWS_CM_90424]	Provide E2E Result
[SWS_CM_90430]	
[SWS_CM_90431]	

Table E.27: Changed Specification Items in 18-03

E.9.3 Deleted Specification Items in 18-03

Number	Heading
[SWS_CM_00121]	Method to find a service
[SWS_CM_00161]	Method to send a service event
[SWS_CM_00163]	Send event where Communication Management is responsible for the data
[SWS_CM_00171]	Receive a service event using polling
[SWS_CM_01014]	No memory allocation in header files
[SWS_CM_01016]	Data Type definitions for AUTOSAR Data Types in Common header file
[SWS_CM_90425]	Namespace of Sample Pointer

Table E.28: Deleted Specification Items in 18-03

E.10 Constraint and Specification Item Changes between AUTOSAR Release R17-03 and R17-10

E.10.1 Added Specification Items in 17-10

Number	Heading
[SWS_CM_00002]	Service skeleton Event class
[SWS_CM_00007]	Service skeleton Field class
[SWS_CM_00112]	Method to get the value of a field
[SWS_CM_00113]	Method to set the value of a field
[SWS_CM_00114]	Registering Getters
[SWS_CM_00115]	Existence of RegisterGetHandler method
[SWS_CM_00116]	Registering Setters
[SWS_CM_00117]	Existence of the RegisterSetHandler method
[SWS_CM_00119]	Update Function
[SWS_CM_00120]	Provision of an update notification event for a Field
[SWS_CM_00128]	Ensuring the existence of valid Field values
[SWS_CM_00129]	Ensuring existence of SetHandler
[SWS_CM_00132]	Existence of getter method
[SWS_CM_00133]	Existence of the set method
[SWS_CM_00182]	Event Receive Handler call serialization
[SWS_CM_00183]	Disable service event trigger
[SWS_CM_00252]	
[SWS_CM_00253]	
[SWS_CM_00254]	
[SWS_CM_00255]	
[SWS_CM_00256]	
[SWS_CM_00257]	
[SWS_CM_00258]	
[SWS_CM_00259]	
[SWS_CM_00260]	
[SWS_CM_00262]	
[SWS_CM_00263]	
[SWS_CM_00264]	
[SWS_CM_00265]	
[SWS_CM_00266]	FilterFunction for incoming event filtering
[SWS_CM_00427]	String Data Type with `baseTypeSize` of 16
[SWS_CM_00428]	Element specification typed by String Data Type with `baseTypeSize` of 16
[SWS_CM_01031]	Service fields namespace

△

Number	Heading
[SWS_CM_10268]	
[SWS_CM_10269]	
[SWS_CM_10270]	
[SWS_CM_10271]	
[SWS_CM_10272]	
[SWS_CM_10273]	
[SWS_CM_10274]	
[SWS_CM_10275]	
[SWS_CM_10276]	
[SWS_CM_10277]	
[SWS_CM_10278]	
[SWS_CM_10279]	
[SWS_CM_10280]	
[SWS_CM_10281]	
[SWS_CM_10282]	
[SWS_CM_10283]	
[SWS_CM_10284]	
[SWS_CM_10285]	Responsibility of proper string encoding
[SWS_CM_10286]	Encoding mismatch in input configurations
[SWS_CM_10287]	Conditions for sending of a SOME/IP event message
[SWS_CM_10288]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10289]	Source of a SOME/IP event message
[SWS_CM_10290]	Destination of a SOME/IP event message
[SWS_CM_10291]	Content of the SOME/IP event message
[SWS_CM_10292]	Checks for a received SOME/IP event message
[SWS_CM_10293]	Identifying the right event
[SWS_CM_10294]	Deserializing the payload
[SWS_CM_10295]	Store the received event data
[SWS_CM_10296]	Invoke receive handler
[SWS_CM_10297]	Conditions for sending of a SOME/IP request message
[SWS_CM_10298]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_10299]	Source of a SOME/IP request message
[SWS_CM_10300]	Destination of a SOME/IP request message
[SWS_CM_10301]	Content of the SOME/IP request message
[SWS_CM_10302]	Checks for a received SOME/IP request message
[SWS_CM_10303]	Identifying the right method
[SWS_CM_10304]	Deserializing the payload
[SWS_CM_10305]	Store the received method data
[SWS_CM_10306]	Invoke the method - event driven

▽

△

Number	Heading
[SWS_CM_10307]	Invoke the method - polling
[SWS_CM_10308]	Conditions for sending of a SOME/IP response message
[SWS_CM_10309]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_10310]	Source of a SOME/IP response message
[SWS_CM_10311]	Destination of a SOME/IP response message
[SWS_CM_10312]	Content of the SOME/IP response message
[SWS_CM_10313]	Checks for a received SOME/IP response message
[SWS_CM_10314]	Identifying the right method
[SWS_CM_10315]	Discarding orphaned responses
[SWS_CM_10316]	Deserializing the payload - response mesages
[SWS_CM_10317]	Making the Future ready
[SWS_CM_10318]	Invoke the notification function
[SWS_CM_10319]	Conditions for sending of a SOME/IP event message
[SWS_CM_10320]	Transport protocol for sending of a SOME/IP event message
[SWS_CM_10321]	Source of a SOME/IP event message
[SWS_CM_10322]	Destination of a SOME/IP event message
[SWS_CM_10323]	Content of the SOME/IP event message
[SWS_CM_10324]	Checks for a received SOME/IP event message
[SWS_CM_10325]	Identifying the right event
[SWS_CM_10326]	Deserializing the payload
[SWS_CM_10327]	Store the received event data
[SWS_CM_10328]	Invoke receive handler
[SWS_CM_10329]	Conditions for sending of a SOME/IP request message
[SWS_CM_10330]	Transport protocol for sending of a SOME/IP request message
[SWS_CM_10331]	Source of a SOME/IP request message
[SWS_CM_10332]	Destination of a SOME/IP request message
[SWS_CM_10333]	Content of the SOME/IP request message
[SWS_CM_10334]	Checks for a received SOME/IP request message
[SWS_CM_10335]	Identifying the right method
[SWS_CM_10336]	Deserializing the payload
[SWS_CM_10337]	Store the received method data
[SWS_CM_10338]	Invoke the registered set/get handlers - event driven
[SWS_CM_10339]	Invoke the registered set/get handlers - polling
[SWS_CM_10340]	Conditions for sending of a SOME/IP response message
[SWS_CM_10341]	Transport protocol for sending of a SOME/IP response message
[SWS_CM_10342]	Source of a SOME/IP response message
[SWS_CM_10343]	Destination of a SOME/IP response message
[SWS_CM_10344]	Content of the SOME/IP response message

▽

△

Number	Heading
[SWS_CM_10345]	Checks for a received SOME/IP response message
[SWS_CM_10346]	Identifying the right method
[SWS_CM_10347]	Discarding orphaned responses
[SWS_CM_10348]	Deserializing the payload
[SWS_CM_10349]	Making the Future ready
[SWS_CM_10350]	Invoke the notification function
[SWS_CM_10351]	Service application errors
[SWS_CM_10352]	Definition of `ServiceNotAvailableException`
[SWS_CM_10353]	Use of `ServiceNotAvailableException`
[SWS_CM_10354]	Definition of `ApplicationErrorException`
[SWS_CM_10355]	Use of `ApplicationErrorException`
[SWS_CM_10356]	Definition of sub-classes of `ApplicationErrorException`
[SWS_CM_10357]	Distinguishing errors from normal responses
[SWS_CM_10358]	Identifying the right application error
[SWS_CM_10359]	Deserializing the payload - error response mesages
[SWS_CM_10361]	
[SWS_CM_10362]	Raising checked exceptions for application errors
[SWS_CM_10370]	Data Type definitions for Application Errors in Common header file
[SWS_CM_10371]	Context of thrown checked exceptions
[SWS_CM_11262]	
[SWS_CM_11263]	
[SWS_CM_90101]	Secure channel creation
[SWS_CM_90102]	Using secure channels
[SWS_CM_90103]	TLS secure channel for methods using reliable transport
[SWS_CM_90104]	DTLS secure channel for methods using unreliable transport
[SWS_CM_90105]	TLS secure channel for events using reliable transport
[SWS_CM_90106]	DTLS secure channel for events using unreliable transport
[SWS_CM_90107]	TLS secure channel for fields
[SWS_CM_90108]	SecOC secure channel for methods
[SWS_CM_90109]	SecOC secure channel for events
[SWS_CM_90110]	SecOC secure channel for fields
[SWS_CM_90401]	
[SWS_CM_90402]	
[SWS_CM_90403]	
[SWS_CM_90404]	
[SWS_CM_90405]	
[SWS_CM_90406]	
[SWS_CM_90407]	
[SWS_CM_90408]	

▽

△

Number	Heading
[SWS_CM_90409]	
[SWS_CM_90410]	
[SWS_CM_90411]	
[SWS_CM_90412]	
[SWS_CM_90413]	
[SWS_CM_90414]	
[SWS_CM_90415]	
[SWS_CM_90416]	
[SWS_CM_90417]	
[SWS_CM_90418]	
[SWS_CM_90419]	
[SWS_CM_90420]	E2ECheckStatus of a sample
[SWS_CM_90421]	ara::com:state_machine::E2E check status
[SWS_CM_90422]	ara::com:state_machine::State
[SWS_CM_90423]	E2EResult
[SWS_CM_90424]	Provide E2E Result
[SWS_CM_90425]	Namespace of Sample Pointer
[SWS_CM_90430]	
[SWS_CM_90431]	
[SWS_CM_90432]	Functionality of Sample Pointer

Table E.29: Added Specification Items in 17-10

E.10.2 Changed Specification Items in 17-10

Number	Heading
[SWS_CM_00122]	Find service with immediately returned request
[SWS_CM_00123]	Find service with handler registration
[SWS_CM_00124]	Find service handler behavior
[SWS_CM_00171]	Receive a service event using polling
[SWS_CM_00181]	Enable service event trigger
[SWS_CM_00195]	Retrieving results of the method call
[SWS_CM_00202]	SOME/IP FindService message
[SWS_CM_00203]	SOME/IP OfferService message
[SWS_CM_00205]	SOME/IP SubscribeEventgroup message
[SWS_CM_00206]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_00300]	Event Cache Update Policy

▽

△

Number	Heading
[SWS_CM_00302]	Instance Identifier Class
[SWS_CM_00303]	Find Service Handle
[SWS_CM_00304]	Service Handle Container
[SWS_CM_00305]	Find Service Handler
[SWS_CM_00306]	Sample Pointer
[SWS_CM_00307]	Sample Container
[SWS_CM_00308]	Sample Allocatee Pointer
[SWS_CM_00309]	Event Receive Handler
[SWS_CM_00310]	Subscription State
[SWS_CM_00312]	Handle Type Class
[SWS_CM_00346]	`Promise::set_value`, forwarding reference version
[SWS_CM_00406]	String Data Type with `baseTypeSize` of 8
[SWS_CM_00409]	Associative Map Data Type
[SWS_CM_00420]	Element specification typed by String Data Type with `baseTypeSize` of 8
[SWS_CM_01010]	Service Identifier and Service Version Classes
[SWS_CM_01016]	Data Type definitions for AUTOSAR Data Types in Common header file
[SWS_CM_01019]	Data Type declarations in Types header file
[SWS_CM_10017]	
[SWS_CM_10034]	
[SWS_CM_10059]	
[SWS_CM_10242]	UTF-8 Strings
[SWS_CM_10243]	UTF-16 Strings
[SWS_CM_10245]	Serialization of strings
[SWS_CM_10247]	Deserialization of strings
[SWS_CM_10252]	
[SWS_CM_10253]	
[SWS_CM_10256]	
[SWS_CM_10257]	
[SWS_CM_10258]	
[SWS_CM_10260]	
[SWS_CM_10262]	Insertion of an associative map length field
[SWS_CM_10264]	Size of the associative map length field
[SWS_CM_10267]	Insertion of an associative map length field

Table E.30: Changed Specification Items in 17-10

E.10.3 Deleted Specification Items in 17-10

Number	Heading
[SWS_CM_01003]	Inclusion protection

Table E.31: Deleted Specification Items in 17-10