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	R21-11

Document Change History

Date	Release	Changed by	Description
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

Table of 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 relevant to the Communication Management that are not included in the AUTOSAR glossary [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

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	Act of connecting a Service Requester to a concrete Service Instance of a Service Provider.
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.

3 Related documentation

3.1 Input documents & related standards and norms

[1] Explanation of ara::com API
AUTOSAR_EXP_ARAComAPI

[2] Glossary
AUTOSAR_TR_Glossary

[3] General Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[4] E2E Protocol Specification
AUTOSAR_PRS_E2EProtocol

[5] SOME/IP Protocol Specification
AUTOSAR_PRS_SOMEIPProtocol

[6] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[7] Requirements on E2E
AUTOSAR_RS_E2E

[8] Requirements on Communication Management
AUTOSAR_RS_CommunicationManagement

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

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

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

[12] SOME/IP Service Discovery Protocol Specification
AUTOSAR_PRS_SOMEIPServiceDiscoveryProtocol

[13] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes

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

[15] UTF-16, an encoding of ISO 10646
http://www.ietf.org/rfc/rfc2781.txt

[16] Specification of Adaptive Platform Core
AUTOSAR_SWS_AdaptivePlatformCore

[17] Specification of Socket Adaptor

AUTOSAR_SWS_SocketAdaptor

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

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

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

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

[22] ISO/IEC C++ 2003 Language DDS PSM, Version 1.0
https://www.omg.org/spec/DDS-PSM-Cxx/1.0

[23] Interface Definition Language (IDL), Version 4.2
https://www.omg.org/spec/IDL/4.2

[24] Specification of Language Binding for modeled AP data types
AUTOSAR_SWS_LanguageBindingForModeledAPdatatypes

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

[26] Specification of Identity and Access Management
AUTOSAR_SWS_IdentityAndAccessManagement

[27] Specification of Secure Onboard Communication Protocol
AUTOSAR_PRS_SecOcProtocol

[28] Integration of DDS Security
AUTOSAR_TR_DDSSecurityIntegration

[29] Methodology for Adaptive Platform
AUTOSAR_TR_AdaptiveMethodology

[30] ISO/IEC 14882:2011, Information technology – Programming languages – C++
http://www.iso.org

[31] N4659: Working Draft, Standard for ProgrammingLanguage C++
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [3, RS General], which is also valid for the CM.

Thus, the specification SWS BSW General shall be considered as additional and required specification for CM.

4 Constraints and assumptions

4.1 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:

- **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.

The general limitations regarding E2E protection and the detectable failure modes are described in [4]. Additional, platform specific limitations regarding E2E protection are described in chapter 7.3.1 and 7.2.1.

The following limitations regarding optionality introduced with the Tag-Length-Value serialization principle described in [5] and [6] apply:

- **Optional method arguments**
  [SWS_CM_CONSTR_00001]{DRAFT} ⌈Communication Management does currently not support the existence of optional method arguments.⌋*()*

In addition the following features are not supported in the current version of this document:

- E2E protection of `ServiceInterface.trigger`s

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other functional clusters

5.1 Platform dependencies

The Communication Management is dependent on the E2E protection protocol defined in [7] and [4]. 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 [8] and links to the fulfilment of these.

Please note that if a requirement contained in [8] is not mentioned in the below table, it means that is not fulfilled by this document.

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_00005] [SWS_CM_00006] [SWS_CM_00007] [SWS_CM_00008] [SWS_CM_00010] [SWS_CM_00011] [SWS_CM_00012] [SWS_CM_00013] [SWS_CM_00014] [SWS_CM_00015] [SWS_CM_00016] [SWS_CM_00017] [SWS_CM_00018] [SWS_CM_00019] [SWS_CM_00020] [SWS_CM_00021] [SWS_CM_00022] [SWS_CM_00023] [SWS_CM_00024] [SWS_CM_00025] [SWS_CM_00026] [SWS_CM_00027] [SWS_CM_00028] [SWS_CM_00029] [SWS_CM_00030] [SWS_CM_00031] [SWS_CM_00032] [SWS_CM_00035] [SWS_CM_00101] [SWS_CM_00111] [SWS_CM_00112] [SWS_CM_00113] [SWS_CM_00114] [SWS_CM_00115] [SWS_CM_00116] [SWS_CM_00117] [SWS_CM_00118] [SWS_CM_00119] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00125] [SWS_CM_00130] [SWS_CM_00131] [SWS_CM_00132] [SWS_CM_00133] [SWS_CM_00134] [SWS_CM_00135] [SWS_CM_00136] [SWS_CM_00137] [SWS_CM_00141] [SWS_CM_00151] [SWS_CM_00152] [SWS_CM_00153] [SWS_CM_00162] [SWS_CM_00181] [SWS_CM_00183] [SWS_CM_00191] [SWS_CM_00192] [SWS_CM_00193] [SWS_CM_00194] [SWS_CM_00195] [SWS_CM_00196] [SWS_CM_00197] [SWS_CM_00198] [SWS_CM_00199] [SWS_CM_00226] [SWS_CM_00227] [SWS_CM_00228] [SWS_CM_00249]

Requirement	Description	Satisfied by
		[SWS_CM_00301] [SWS_CM_00302] [SWS_CM_00304] [SWS_CM_00306] [SWS_CM_00308] [SWS_CM_00309] [SWS_CM_00310] [SWS_CM_00311] [SWS_CM_00312] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_00316] [SWS_CM_00317] [SWS_CM_00318] [SWS_CM_00319] [SWS_CM_00333] [SWS_CM_00334] [SWS_CM_00351] [SWS_CM_00383] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00700] [SWS_CM_00701] [SWS_CM_00702] [SWS_CM_00703] [SWS_CM_00704] [SWS_CM_00705] [SWS_CM_00706] [SWS_CM_00707] [SWS_CM_00714] [SWS_CM_00721] [SWS_CM_00722] [SWS_CM_00723] [SWS_CM_00724] [SWS_CM_00810] [SWS_CM_01001] [SWS_CM_01002] [SWS_CM_01004] [SWS_CM_01005] [SWS_CM_01006] [SWS_CM_01007] [SWS_CM_01009] [SWS_CM_01012] [SWS_CM_01013] [SWS_CM_01015] [SWS_CM_01018] [SWS_CM_01020] [SWS_CM_01031] [SWS_CM_01050] [SWS_CM_01051] [SWS_CM_01052] [SWS_CM_01053] [SWS_CM_01054] [SWS_CM_01055] [SWS_CM_01056] [SWS_CM_01057] [SWS_CM_01058] [SWS_CM_01059] [SWS_CM_01060] [SWS_CM_01061] [SWS_CM_01062] [SWS_CM_01063] [SWS_CM_01064] [SWS_CM_01065] [SWS_CM_01066] [SWS_CM_01067] [SWS_CM_01068] [SWS_CM_01069] [SWS_CM_10362] [SWS_CM_10372] [SWS_CM_10383] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_10440] [SWS_CM_10446] [SWS_CM_11251] [SWS_CM_11266] [SWS_CM_11326] [SWS_CM_11350] [SWS_CM_11351] [SWS_CM_11352] [SWS_CM_11353] [SWS_CM_11354] [SWS_CM_11355] [SWS_CM_11356] [SWS_CM_11357] [SWS_CM_11358] [SWS_CM_11359] [SWS_CM_11360] [SWS_CM_11361] [SWS_CM_11362] [SWS_CM_11363] [SWS_CM_11370] [SWS_CM_11371] [SWS_CM_12000] [SWS_CM_90420] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90424] [SWS_CM_90426] [SWS_CM_90427] [SWS_CM_90434] [SWS_CM_90435] [SWS_CM_90437] [SWS_CM_90438]

Requirement	Description	Satisfied by
[RS_AP_00115]	Public namespaces.	[SWS_CM_00013] [SWS_CM_00018] [SWS_CM_00019] [SWS_CM_00024] [SWS_CM_00118] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00152] [SWS_CM_00153] [SWS_CM_00198] [SWS_CM_00316] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_11264] [SWS_CM_11352] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90424] [SWS_CM_90426] [SWS_CM_90427] [SWS_CM_90438]
[RS_AP_00116]	Header file name.	[SWS_CM_01002] [SWS_CM_01012] [SWS_CM_01013]
[RS_AP_00119]	Return values / application errors.	[SWS_CM_00018] [SWS_CM_00019] [SWS_CM_00021] [SWS_CM_00024] [SWS_CM_00118] [SWS_CM_00122] [SWS_CM_00123] [SWS_CM_00228] [SWS_CM_00310] [SWS_CM_00316] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00704] [SWS_CM_00706] [SWS_CM_10362] [SWS_CM_10383] [SWS_CM_10440] [SWS_CM_11264] [SWS_CM_11265] [SWS_CM_11266] [SWS_CM_11351] [SWS_CM_11352] [SWS_CM_11353] [SWS_CM_11355] [SWS_CM_11357] [SWS_CM_11359] [SWS_CM_11361] [SWS_CM_11363] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90426] [SWS_CM_90427]
[RS_AP_00120]	Method and Function names.	[SWS_CM_00010] [SWS_CM_00011] [SWS_CM_00012] [SWS_CM_00013] [SWS_CM_00014] [SWS_CM_00015] [SWS_CM_00016] [SWS_CM_00020] [SWS_CM_00022] [SWS_CM_00023] [SWS_CM_00024] [SWS_CM_00025] [SWS_CM_00026] [SWS_CM_00027] [SWS_CM_00028] [SWS_CM_00029] [SWS_CM_00030] [SWS_CM_00031] [SWS_CM_00032] [SWS_CM_00035] [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_00141] [SWS_CM_00151] [SWS_CM_00162] [SWS_CM_00181] [SWS_CM_00183] [SWS_CM_00192] [SWS_CM_00195] [SWS_CM_00196] [SWS_CM_00198] [SWS_CM_00199] [SWS_CM_00226] [SWS_CM_00249] [SWS_CM_00309] [SWS_CM_00311] [SWS_CM_00316] [SWS_CM_00333] [SWS_CM_00334]

Requirement	Description	Satisfied by
		[SWS_CM_00351] [SWS_CM_00383] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_00721] [SWS_CM_00722] [SWS_CM_00723] [SWS_CM_00724] [SWS_CM_00810] [SWS_CM_11292] [SWS_CM_11295] [SWS_CM_11296] [SWS_CM_11297] [SWS_CM_11298] [SWS_CM_11299] [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_11350] [SWS_CM_11352] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11358] [SWS_CM_11360] [SWS_CM_11362] [SWS_CM_90420] [SWS_CM_90435] [SWS_CM_90437] [SWS_CM_90438]
[RS_AP_00121]	Parameter names.	[SWS_CM_00012] [SWS_CM_00016] [SWS_CM_00018] [SWS_CM_00019] [SWS_CM_00020] [SWS_CM_00025] [SWS_CM_00028] [SWS_CM_00031] [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_00249] [SWS_CM_00333] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00701] [SWS_CM_00721] [SWS_CM_00722] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_11292] [SWS_CM_11296] [SWS_CM_11297] [SWS_CM_11299] [SWS_CM_11328] [SWS_CM_11332] [SWS_CM_11333] [SWS_CM_11335] [SWS_CM_11352] [SWS_CM_90437]
[RS_AP_00122]	Type names.	[SWS_CM_00002] [SWS_CM_00004] [SWS_CM_00302] [SWS_CM_00303] [SWS_CM_00304] [SWS_CM_00306] [SWS_CM_00308] [SWS_CM_00312] [SWS_CM_00319] [SWS_CM_01050] [SWS_CM_10432] [SWS_CM_11291] [SWS_CM_11293] [SWS_CM_11327] [SWS_CM_11329] [SWS_CM_12367]
[RS_AP_00125]	Enumerator and constant names.	[SWS_CM_00301] [SWS_CM_00310]

Requirement	Description	Satisfied by
[RS_AP_00127]	Usage of ara::core types.	[SWS_CM_00014] [SWS_CM_00015] [SWS_CM_00017] [SWS_CM_00018] [SWS_CM_00019] [SWS_CM_00027] [SWS_CM_00030] [SWS_CM_00031] [SWS_CM_00032] [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_00228] [SWS_CM_00302] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_00701] [SWS_CM_00704] [SWS_CM_00705] [SWS_CM_00706] [SWS_CM_01050] [SWS_CM_10362] [SWS_CM_10432] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_10440] [SWS_CM_10446] [SWS_CM_11266] [SWS_CM_11291] [SWS_CM_11293] [SWS_CM_11326] [SWS_CM_11327] [SWS_CM_11329] [SWS_CM_11350] [SWS_CM_11351] [SWS_CM_11353] [SWS_CM_11354] [SWS_CM_11355] [SWS_CM_11356] [SWS_CM_11357] [SWS_CM_11358] [SWS_CM_11359] [SWS_CM_11360] [SWS_CM_11361] [SWS_CM_11362] [SWS_CM_11363] [SWS_CM_12367]
[RS_AP_00128]	Error reporting.	[SWS_CM_00014] [SWS_CM_00015] [SWS_CM_00017] [SWS_CM_00027] [SWS_CM_00030] [SWS_CM_00032] [SWS_CM_00112] [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_00701] [SWS_CM_00705] [SWS_CM_00706] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_11326] [SWS_CM_11350] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11358] [SWS_CM_11360] [SWS_CM_11362]

Requirement	Description	Satisfied by
[RS_AP_00130]	AUTOSAR Adaptive Platform shall represent a rich and modern programming environment.	[SWS_CM_10432] [SWS_CM_10474] [SWS_CM_11267] [SWS_CM_11268] [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_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] [SWS_CM_11341] [SWS_CM_11342] [SWS_CM_12367] [SWS_CM_99023] [SWS_CM_99024] [SWS_CM_99025] [SWS_CM_99026] [SWS_CM_99027]
[RS_AP_00132]	noexcept behavior of API functions	[SWS_CM_00027] [SWS_CM_00306] [SWS_CM_00705] [SWS_CM_01050] [SWS_CM_01052] [SWS_CM_01054] [SWS_CM_01055] [SWS_CM_01060] [SWS_CM_01062] [SWS_CM_01064] [SWS_CM_01065] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_11292] [SWS_CM_11295] [SWS_CM_11296] [SWS_CM_11298] [SWS_CM_11299] [SWS_CM_11326] [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_90420]
[RS_AP_00134]	noexcept behavior of class destructors	[SWS_CM_01050] [SWS_CM_01059]
[RS_AP_00135]	Avoidance of shared ownership.	[SWS_CM_00306] [SWS_CM_00308]
[RS_AP_00136]	Usage of string types.	[SWS_CM_10054] [SWS_CM_10242] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10285] [SWS_CM_11046]
[RS_AP_00137]	Connecting run-time interface with model.	[SWS_CM_00018] [SWS_CM_00019] [SWS_CM_00118] [SWS_CM_00152] [SWS_CM_00622] [SWS_CM_00623] [SWS_CM_10436] [SWS_CM_10450] [SWS_CM_10452] [SWS_CM_10590]
[RS_AP_00138]	Return type of asynchronous function calls.	[SWS_CM_00014] [SWS_CM_00015] [SWS_CM_00017] [SWS_CM_00017] [SWS_CM_00030] [SWS_CM_00030] [SWS_CM_00031] [SWS_CM_00031] [SWS_CM_00031] [SWS_CM_00032] [SWS_CM_00032] [SWS_CM_00112] [SWS_CM_00112] [SWS_CM_00113] [SWS_CM_00113] [SWS_CM_00113] [SWS_CM_00114] [SWS_CM_00116] [SWS_CM_00191] [SWS_CM_00191] [SWS_CM_00192] [SWS_CM_00192] [SWS_CM_00195] [SWS_CM_00196]

Requirement	Description	Satisfied by
		[SWS_CM_00196] [SWS_CM_00197] [SWS_CM_00199] [SWS_CM_00199] [SWS_CM_10414] [SWS_CM_11350] [SWS_CM_11350] [SWS_CM_11354] [SWS_CM_11354] [SWS_CM_11356] [SWS_CM_11356] [SWS_CM_11358] [SWS_CM_11358] [SWS_CM_11360] [SWS_CM_11360] [SWS_CM_11362] [SWS_CM_11362]
[RS_AP_00139]	Return type of synchronous function calls.	[SWS_CM_00027] [SWS_CM_00027] [SWS_CM_00195] [SWS_CM_00226] [SWS_CM_00226] [SWS_CM_00701] [SWS_CM_00701] [SWS_CM_00705] [SWS_CM_00705] [SWS_CM_00706] [SWS_CM_10435] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_10438] [SWS_CM_11326] [SWS_CM_11326]
[RS_AP_00145]	Availability of special member functions.	[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_00306] [SWS_CM_00317] [SWS_CM_00318] [SWS_CM_01050] [SWS_CM_01051] [SWS_CM_01052] [SWS_CM_01053] [SWS_CM_01054] [SWS_CM_01055] [SWS_CM_01056] [SWS_CM_01057] [SWS_CM_01058] [SWS_CM_01059] [SWS_CM_01060] [SWS_CM_01061] [SWS_CM_10435] [SWS_CM_10436] [SWS_CM_10437] [SWS_CM_10438] [SWS_CM_10446] [SWS_CM_10482] [SWS_CM_10483] [SWS_CM_11303] [SWS_CM_11304] [SWS_CM_11305] [SWS_CM_11306] [SWS_CM_11312] [SWS_CM_11313] [SWS_CM_11314] [SWS_CM_11315] [SWS_CM_11316] [SWS_CM_11317] [SWS_CM_11370] [SWS_CM_11371]
[RS_AP_00147]	Classes which are created by an InstanceSpecifer shall not be copyable, but at most movable.	[SWS_CM_00134] [SWS_CM_00135] [SWS_CM_00136] [SWS_CM_00137] [SWS_CM_11303] [SWS_CM_11304] [SWS_CM_11305] [SWS_CM_11306] [SWS_CM_11316] [SWS_CM_11317]
[RS_CM_00001]	The Communication Management shall provide a standardized header file structure for each service.	[SWS_CM_01001] [SWS_CM_01002] [SWS_CM_01004] [SWS_CM_01012] [SWS_CM_01013] [SWS_CM_01017] [SWS_CM_01019] [SWS_CM_01020] [SWS_CM_10370] [SWS_CM_10372] [SWS_CM_10453] [SWS_CM_10488] [SWS_CM_10490] [SWS_CM_12000]

Requirement	Description	Satisfied by
[RS_CM_00002]	The service header files shall define the namespace for the respective service.	[SWS_CM_01005] [SWS_CM_01006] [SWS_CM_01007] [SWS_CM_01008] [SWS_CM_01009] [SWS_CM_01015] [SWS_CM_01018] [SWS_CM_01031] [SWS_CM_10489]
[RS_CM_00004]	Communication Management shall support the translation between signal-based and service-oriented communication	[SWS_CM_10360] [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_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]
[RS_CM_00101]	Communication Management shall provide an interface to offer services	[SWS_CM_00002] [SWS_CM_00010] [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_10435] [SWS_CM_10436] [SWS_CM_10437] [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_11326] [SWS_CM_90500] [SWS_CM_90502] [SWS_CM_90503] [SWS_CM_90504] [SWS_CM_90505] [SWS_CM_90506] [SWS_CM_90507] [SWS_CM_90508]

Requirement	Description	Satisfied by
[RS_CM_00102]	Communication Management shall provide an interface to find services	[SWS_CM_00004] [SWS_CM_00018] [SWS_CM_00019] [SWS_CM_00020] [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_11264] [SWS_CM_11352] [SWS_CM_90500] [SWS_CM_90510] [SWS_CM_90511] [SWS_CM_90512] [SWS_CM_90513] [SWS_CM_90514]
[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_00022] [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_00724] [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]
[RS_CM_00104]	Communication Management shall provide an interface to stop the subscription to an event of a service instance	[SWS_CM_00023] [SWS_CM_00035] [SWS_CM_00151] [SWS_CM_00207] [SWS_CM_00310] [SWS_CM_00311] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_00810] [SWS_CM_10378] [SWS_CM_10530] [SWS_CM_11021] [SWS_CM_11136]
[RS_CM_00105]	Communication Management shall provide an interface to stop offering services	[SWS_CM_00011] [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_00024] [SWS_CM_00025] [SWS_CM_00026] [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]

Requirement	Description	Satisfied by
[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_00021] [SWS_CM_00313] [SWS_CM_00314] [SWS_CM_00315] [SWS_CM_10383] [SWS_CM_10491]
[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_00102] [SWS_CM_00118] [SWS_CM_00202] [SWS_CM_00203] [SWS_CM_00205] [SWS_CM_01010] [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] [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_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]

Requirement	Description	Satisfied by
[RS_CM_00201]	Communication Management shall provide an API to send events to other applications	[SWS_CM_00003] [SWS_CM_00012] [SWS_CM_00013] [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_00722] [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_10226] [SWS_CM_10227] [SWS_CM_10230] [SWS_CM_10234] [SWS_CM_10235] [SWS_CM_10242] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10248] [SWS_CM_10250] [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] [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]

Requirement	Description	Satisfied by
		[SWS_CM_10323] [SWS_CM_10324] [SWS_CM_10325] [SWS_CM_10326] [SWS_CM_10360] [SWS_CM_10361] [SWS_CM_10363] [SWS_CM_10391] [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_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]
[RS_CM_00202]	Communication Management shall provide an API to the application to poll received events	[SWS_CM_00027] [SWS_CM_00226] [SWS_CM_00227] [SWS_CM_00228] [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_00702] [SWS_CM_00703] [SWS_CM_00704] [SWS_CM_00705] [SWS_CM_00706] [SWS_CM_00707] [SWS_CM_00714] [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] [SWS_CM_10098] [SWS_CM_10099] [SWS_CM_10169] [SWS_CM_10218] [SWS_CM_10219] [SWS_CM_10222] [SWS_CM_10226]

Requirement	Description	Satisfied by
		[SWS_CM_10227] [SWS_CM_10230] [SWS_CM_10234] [SWS_CM_10235] [SWS_CM_10242] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10248] [SWS_CM_10250] [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_10391] [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_11251] [SWS_CM_11262] [SWS_CM_11263] [SWS_CM_80102] [SWS_CM_80103]
[RS_CM_00203]	Communication Management shall trigger the application on reception of an event	[SWS_CM_00028] [SWS_CM_00029] [SWS_CM_00181] [SWS_CM_00182] [SWS_CM_00183] [SWS_CM_00249] [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_80030] [SWS_CM_80072]

Requirement	Description	Satisfied by
[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_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_00252] [SWS_CM_00253] [SWS_CM_00254] [SWS_CM_00255] [SWS_CM_00256] [SWS_CM_00257] [SWS_CM_00258] [SWS_CM_00259] [SWS_CM_00264] [SWS_CM_01046] [SWS_CM_09004] [SWS_CM_10000] [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_10174] [SWS_CM_10218] [SWS_CM_10219] [SWS_CM_10222] [SWS_CM_10230] [SWS_CM_10234] [SWS_CM_10235] [SWS_CM_10240] [SWS_CM_10242] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10248] [SWS_CM_10252] [SWS_CM_10253] [SWS_CM_10255] [SWS_CM_10256] [SWS_CM_10257] [SWS_CM_10258] [SWS_CM_10259] [SWS_CM_10260] [SWS_CM_10260] [SWS_CM_10261] [SWS_CM_10262] [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_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]

Requirement	Description	Satisfied by
		[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_10360] [SWS_CM_10361] [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] [SWS_CM_10390] [SWS_CM_10391] [SWS_CM_10429] [SWS_CM_10430] [SWS_CM_10431] [SWS_CM_10441] [SWS_CM_10442] [SWS_CM_10444] [SWS_CM_10447] [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_10518] [SWS_CM_10519] [SWS_CM_10520] [SWS_CM_10521] [SWS_CM_10522] [SWS_CM_10523] [SWS_CM_10524] [SWS_CM_10525] [SWS_CM_10526] [SWS_CM_10527] [SWS_CM_10528] [SWS_CM_10529] [SWS_CM_10530] [SWS_CM_10531]

Requirement	Description	Satisfied by
		[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_11262] [SWS_CM_11263] [SWS_CM_11269] [SWS_CM_11364] [SWS_CM_80001] [SWS_CM_80003] [SWS_CM_80017] [SWS_CM_80019] [SWS_CM_80020]

Requirement	Description	Satisfied by
		[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] [SWS_CM_80102] [SWS_CM_80103] [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_80512] [SWS_CM_80513] [SWS_CM_90443] [SWS_CM_90444] [SWS_CM_90445] [SWS_CM_90446] [SWS_CM_90451] [SWS_CM_90452] [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]
[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_01046] [SWS_CM_01050] [SWS_CM_01051] [SWS_CM_01052] [SWS_CM_01053] [SWS_CM_01054] [SWS_CM_01055] [SWS_CM_01056] [SWS_CM_01057] [SWS_CM_01058] [SWS_CM_01059] [SWS_CM_01060] [SWS_CM_01061] [SWS_CM_01062] [SWS_CM_01063] [SWS_CM_01064] [SWS_CM_01065] [SWS_CM_01066] [SWS_CM_01067] [SWS_CM_01068] [SWS_CM_01069] [SWS_CM_10000] [SWS_CM_80001]
[RS_CM_00211]	Communication Management shall provide an interface to provide methods to other applications	[SWS_CM_00017] [SWS_CM_00191] [SWS_CM_00198] [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_10036] [SWS_CM_10037] [SWS_CM_10042] [SWS_CM_10053] [SWS_CM_10054] [SWS_CM_10055] [SWS_CM_10056] [SWS_CM_10057]

Requirement	Description	Satisfied by
		[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_10226] [SWS_CM_10227] [SWS_CM_10230] [SWS_CM_10234] [SWS_CM_10235] [SWS_CM_10242] [SWS_CM_10245] [SWS_CM_10247] [SWS_CM_10248] [SWS_CM_10250] [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] [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_10362] [SWS_CM_10371] [SWS_CM_10391] [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_11266] [SWS_CM_11350] [SWS_CM_11351] [SWS_CM_11353] [SWS_CM_11354] [SWS_CM_11355] [SWS_CM_11356] [SWS_CM_11357] [SWS_CM_11358] [SWS_CM_11359] [SWS_CM_11360] [SWS_CM_11361] [SWS_CM_11362] [SWS_CM_11363] [SWS_CM_90501]

Requirement	Description	Satisfied by
[RS_CM_00212]	Communication Management shall provide an interface to call methods of other applications synchronously	[SWS_CM_00006] [SWS_CM_00032] [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_10362] [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] [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_11351] [SWS_CM_11353] [SWS_CM_11355] [SWS_CM_11357] [SWS_CM_11359] [SWS_CM_11361] [SWS_CM_11363]

Requirement	Description	Satisfied by
[RS_CM_00213]	Communication Management shall provide an interface to call service methods asynchronously	[SWS_CM_00006] [SWS_CM_00032] [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_10362] [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_11351] [SWS_CM_11353] [SWS_CM_11355] [SWS_CM_11357] [SWS_CM_11359] [SWS_CM_11361] [SWS_CM_11363]
[RS_CM_00214]	Communication Management shall provide an interface to query the result of an asynchronously called service method	[SWS_CM_00193] [SWS_CM_10362] [SWS_CM_10371] [SWS_CM_10440] [SWS_CM_11351] [SWS_CM_11353] [SWS_CM_11355] [SWS_CM_11357] [SWS_CM_11359] [SWS_CM_11361] [SWS_CM_11363]
[RS_CM_00215]	Communication Management shall trigger the application on completion of an asynchronously called service method	[SWS_CM_00197] [SWS_CM_10317] [SWS_CM_10318] [SWS_CM_10349] [SWS_CM_10350] [SWS_CM_11104] [SWS_CM_11108] [SWS_CM_11148]

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_01031] [SWS_CM_90501]
[RS_CM_00217]	Communication Management shall provide a method to remotely set the field value	[SWS_CM_00031] [SWS_CM_00113] [SWS_CM_10329] [SWS_CM_10333] [SWS_CM_10335] [SWS_CM_10344] [SWS_CM_10346] [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_00014] [SWS_CM_00015] [SWS_CM_00016] [SWS_CM_00030] [SWS_CM_00112] [SWS_CM_00114] [SWS_CM_00115] [SWS_CM_00116] [SWS_CM_00117] [SWS_CM_00119] [SWS_CM_00120] [SWS_CM_00128] [SWS_CM_00129] [SWS_CM_00132] [SWS_CM_00133] [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_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]
[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_10471] [SWS_CM_10472] [SWS_CM_10473] [SWS_CM_90406] [SWS_CM_90430] [SWS_CM_90433] [SWS_CM_90453] [SWS_CM_90485]
[RS_CM_00224]	The communication management shall provide the E2E information of the received event to the application.	[SWS_CM_10475] [SWS_CM_90407] [SWS_CM_90408] [SWS_CM_90412] [SWS_CM_90413] [SWS_CM_90417] [SWS_CM_90431] [SWS_CM_90457]
[RS_CM_00225]	Communication Management shall provide an interface to call fire&forget service methods	[SWS_CM_90434] [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_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_90458] [SWS_CM_90459] [SWS_CM_90460] [SWS_CM_90462] [SWS_CM_90463] [SWS_CM_90466] [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_10470] [SWS_CM_10471] [SWS_CM_90464] [SWS_CM_90465] [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_10467] [SWS_CM_10470] [SWS_CM_10471] [SWS_CM_90464] [SWS_CM_90465]
[RS_CM_00410]	The Communication Management shall provide an API to support reading and writing raw data streams that has no datatype information	[SWS_CM_10476] [SWS_CM_10477] [SWS_CM_10478] [SWS_CM_10479] [SWS_CM_10480] [SWS_CM_10481] [SWS_CM_10482] [SWS_CM_10483] [SWS_CM_10484] [SWS_CM_10485] [SWS_CM_10486] [SWS_CM_10487] [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]

Requirement	Description	Satisfied by
		[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_90216] [SWS_CM_90217] [SWS_CM_99004] [SWS_CM_99006]
[RS_CM_00411]	Application developers shall be able to send and receive raw binary data streams independent of the underlying network protocol	[SWS_CM_10476] [SWS_CM_10477] [SWS_CM_10478] [SWS_CM_10479] [SWS_CM_10480] [SWS_CM_10481] [SWS_CM_10482] [SWS_CM_10483] [SWS_CM_10484] [SWS_CM_10485] [SWS_CM_10486] [SWS_CM_10487] [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_90216] [SWS_CM_90217] [SWS_CM_99004] [SWS_CM_99005] [SWS_CM_99006]
[RS_CM_00412]	The Communication Management shall provide TCP/IP Sockets as network protocol for Raw Data Streams	[SWS_CM_10476] [SWS_CM_10477] [SWS_CM_10478] [SWS_CM_10479] [SWS_CM_10480] [SWS_CM_10482] [SWS_CM_10483] [SWS_CM_10484] [SWS_CM_10485] [SWS_CM_10486] [SWS_CM_10487] [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_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_90216] [SWS_CM_99004] [SWS_CM_99005]
[RS_CM_00500]	Service Contract Version for a Service Interface	[SWS_CM_01010] [SWS_CM_09004] [SWS_CM_90508] [SWS_CM_99003]
[RS_CM_00501]	Service Contract Versioning for all Transport Deployment Protocols	[SWS_CM_09004] [SWS_CM_90508] [SWS_CM_99003]

Requirement	Description	Satisfied by
[RS_CM_00600]	Creation of Communication Groups	[SWS_CM_99000] [SWS_CM_99001] [SWS_CM_99002] [SWS_CM_99007] [SWS_CM_99008] [SWS_CM_99009] [SWS_CM_99010] [SWS_CM_99011] [SWS_CM_99012] [SWS_CM_99013] [SWS_CM_99014] [SWS_CM_99015] [SWS_CM_99016] [SWS_CM_99017] [SWS_CM_99018] [SWS_CM_99019] [SWS_CM_99020] [SWS_CM_99021] [SWS_CM_99022]
[RS_CM_00601]	Provide origin of information	[SWS_CM_99000] [SWS_CM_99001] [SWS_CM_99002] [SWS_CM_99007] [SWS_CM_99008] [SWS_CM_99009] [SWS_CM_99010] [SWS_CM_99011] [SWS_CM_99012] [SWS_CM_99013] [SWS_CM_99014] [SWS_CM_99015] [SWS_CM_99016] [SWS_CM_99017] [SWS_CM_99018] [SWS_CM_99019] [SWS_CM_99020] [SWS_CM_99021] [SWS_CM_99022]
[RS_CM_00700]	The Service Discovery shall evaluate the service version compatibility for service connection	[SWS_CM_99003]
[RS_CM_00701]	Service Versioning Blacklist	[SWS_CM_10202]
[RS_E2E_08534]	E2E protocol shall provide E2E Check status to the application	[SWS_CM_10475] [SWS_CM_90411] [SWS_CM_90413] [SWS_CM_90416] [SWS_CM_90417] [SWS_CM_90420] [SWS_CM_90421] [SWS_CM_90422] [SWS_CM_90424] [SWS_CM_90426] [SWS_CM_90427] [SWS_CM_90431] [SWS_CM_90461] [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_10460] [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] [SWS_CM_90453] [SWS_CM_90454] [SWS_CM_90455] [SWS_CM_90456] [SWS_CM_90457]

Requirement	Description	Satisfied by
[RS_E2E_08541]	E2E protocol shall support protected non-periodic communication	[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_90458] [SWS_CM_90459] [SWS_CM_90460] [SWS_CM_90461] [SWS_CM_90462] [SWS_CM_90463] [SWS_CM_90466] [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_10499] [SWS_CM_10500] [SWS_CM_10501] [SWS_CM_10502] [SWS_CM_10503] [SWS_CM_10504] [SWS_CM_10505] [SWS_CM_10506] [SWS_CM_10507] [SWS_CM_10540] [SWS_CM_10541] [SWS_CM_90218]
[RS_IAM_00002]	Position of Policy Enforcement	[SWS_CM_10492] [SWS_CM_10493] [SWS_CM_10494] [SWS_CM_10498] [SWS_CM_10499] [SWS_CM_10500] [SWS_CM_10501] [SWS_CM_10502] [SWS_CM_10503] [SWS_CM_10504] [SWS_CM_10505] [SWS_CM_10506] [SWS_CM_10507] [SWS_CM_10540] [SWS_CM_10541] [SWS_CM_90218]
[RS_IAM_00006]	Access control policies shall be available to the PDP	[SWS_CM_10538] [SWS_CM_10539] [SWS_CM_90001] [SWS_CM_90002] [SWS_CM_90003] [SWS_CM_90005] [SWS_CM_90006] [SWS_CM_90007]
[RS_IAM_00007]	The `Adaptive Platform Foundation` shall provide access control decisions	[SWS_CM_10538] [SWS_CM_10539] [SWS_CM_90001] [SWS_CM_90002] [SWS_CM_90003] [SWS_CM_90005] [SWS_CM_90006] [SWS_CM_90007]
[RS_IAM_00010]	Adaptive applications shall only be able to use AUTOSAR Resources when authorized	[SWS_CM_10538] [SWS_CM_10539] [SWS_CM_90001] [SWS_CM_90002] [SWS_CM_90003] [SWS_CM_90005] [SWS_CM_90006] [SWS_CM_90007]

Requirement	Description	Satisfied by
[RS_SEC_04001]	Secure communication shall be transmitted using secure channels	[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_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] [SWS_CM_90211] [SWS_CM_90212] [SWS_CM_90213] [SWS_CM_90214] [SWS_CM_90215]
[RS_SEC_04002]	Secure channels shall be configurable	[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]
[RS_SEC_04003]	The assignment of communication to specific secure channels shall be configurable	[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_90102] [SWS_CM_90202] [SWS_CM_90212]
[RS_SEC_04004]	Using secure channels shall be transparent on the communication API	[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_90111] [SWS_CM_90112] [SWS_CM_90113] [SWS_CM_90114] [SWS_CM_90119]
[RS_SOMEIPSD_-00002]	SOME/IP Service Discovery Protocol shall support unicast messages	[SWS_CM_00206]

Requirement	Description	Satisfied by
[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_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_-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]
[RS_SOMEIP_00003]	SOME/IP protocol shall provide support of multiple versions of a service interface	[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]

Requirement	Description	Satisfied by
[RS_SOMEIP_00004]	SOME/IP protocol shall support event communication	[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_10360] [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_10034] [SWS_CM_10287] [SWS_CM_10319] [SWS_CM_10360] [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]
[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]

Requirement	Description	Satisfied by
		[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_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_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_80022] [SWS_CM_80064]
[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]

Requirement	Description	Satisfied by
[RS_SOMEIP_00014]	SOME/IP protocol shall support handling of protocol errors on receiver side	[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_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_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_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_10013] [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_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]

Requirement	Description	Satisfied by
[RS_SOMEIP_00042]	SOME/IP protocol shall support unicast and multicast based event communication	[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] [SWS_CM_01050] [SWS_CM_01051] [SWS_CM_01052] [SWS_CM_01053] [SWS_CM_01054] [SWS_CM_01055] [SWS_CM_01056] [SWS_CM_01057] [SWS_CM_01058] [SWS_CM_01059] [SWS_CM_01060] [SWS_CM_01061] [SWS_CM_01062] [SWS_CM_01063] [SWS_CM_01064] [SWS_CM_01065] [SWS_CM_01066] [SWS_CM_01067] [SWS_CM_01068] [SWS_CM_01069]
[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]

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

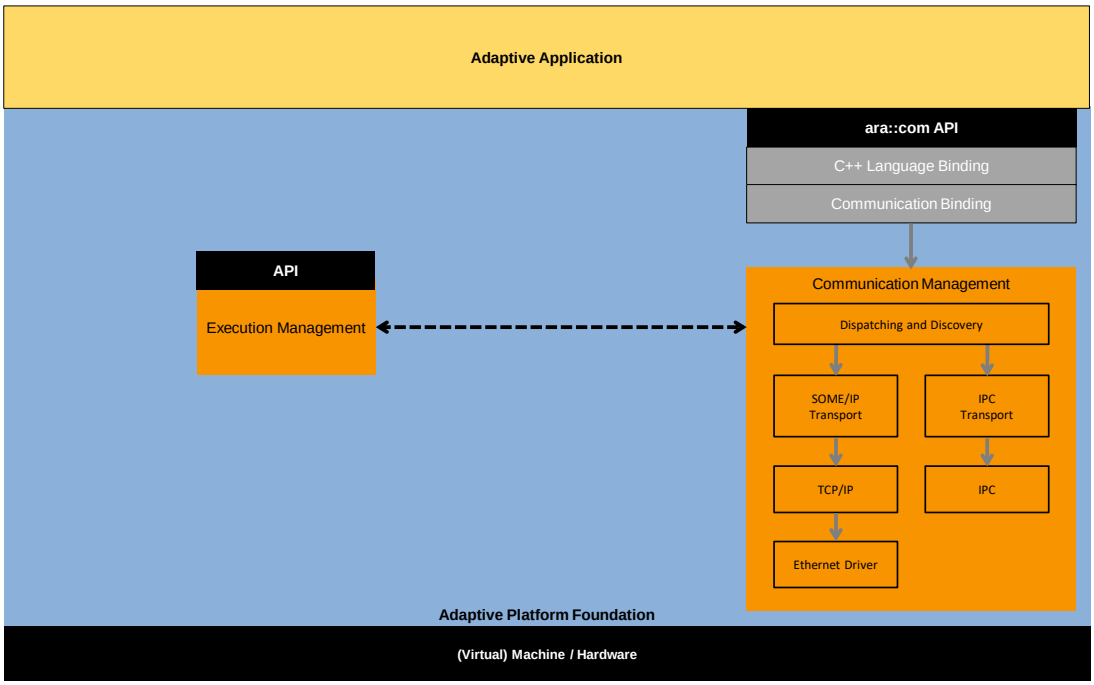


Figure 7.1: Technical Architecture of Communication Management

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 [9] [10].

- 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 [7] and [4].
- 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

Service-Oriented Communication (SoC) as a part of Service-Oriented Architecture (SOA) [11] 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.

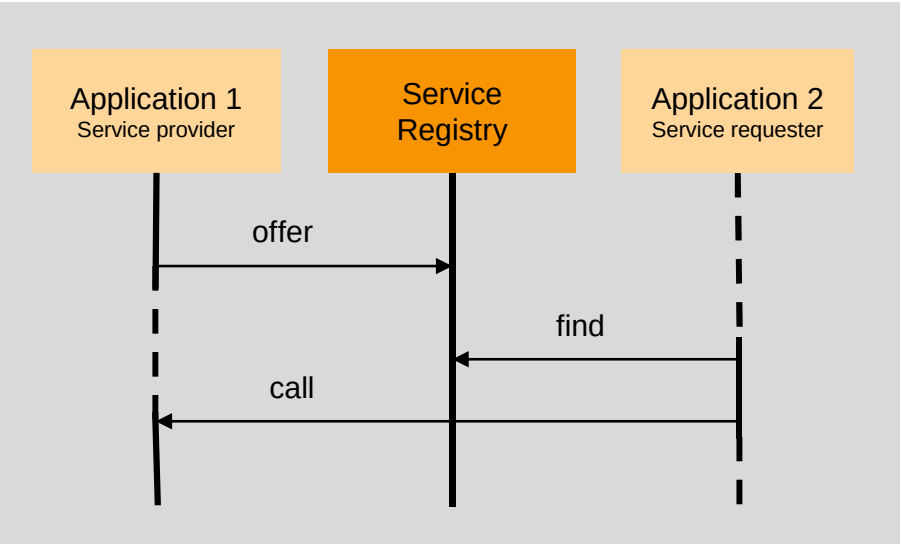


Figure 7.2: Service-Oriented Communication

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* [12].

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.

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.

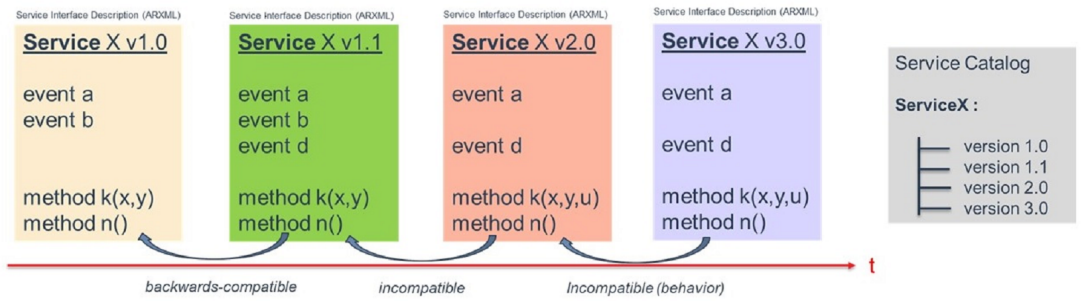


Figure 7.3: Service contract versioning over time

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.

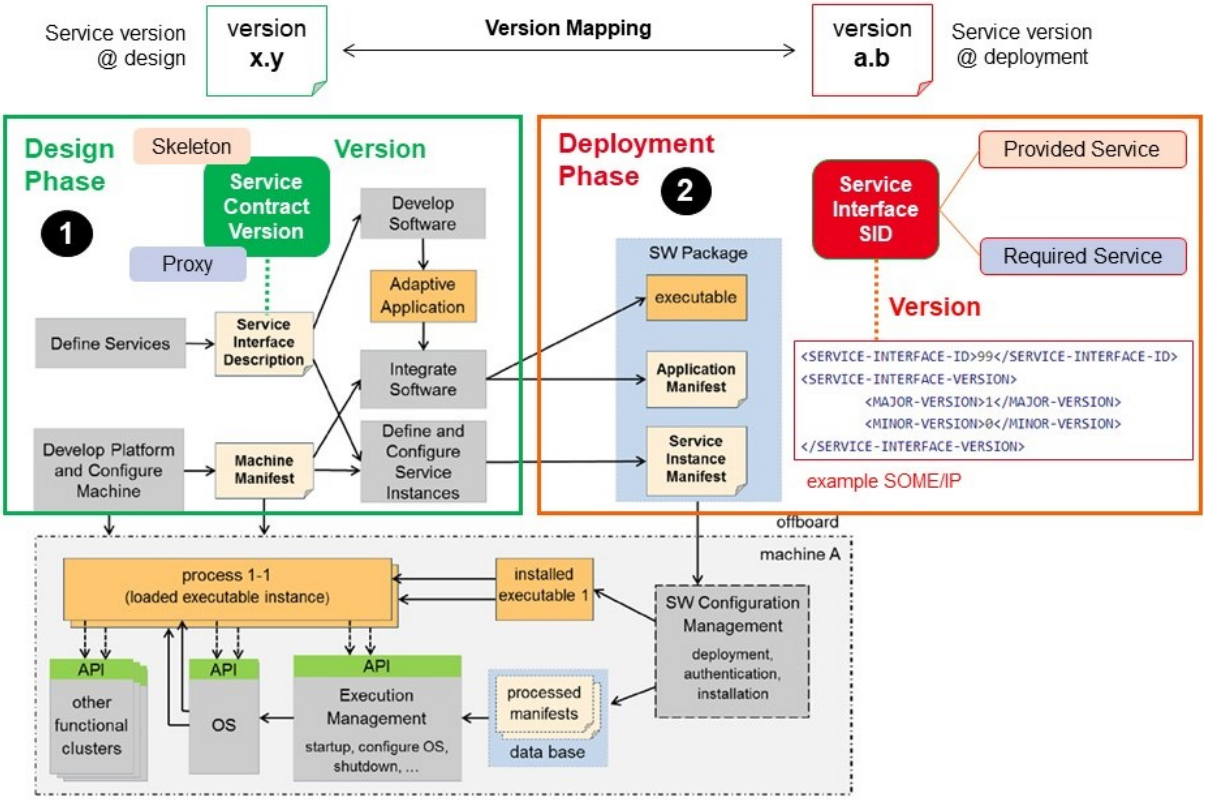


Figure 7.4: Service contract versioning flow

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 ServiceInterfaceDeployments for the same ServiceInterface. Such a mapping will result in unambiguous identification on each VLAN according to the [constr_1723] in [6].

[SWS_CM_99003]{DRAFT} ⌈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.⌋*(RS_CM_00500, RS_CM_00501, RS_CM_00700)*

7.2 End-to-end communication protection for Events

This section specifies the integration of E2E communication protection in ara::com for the processing of Events.

[SWS_CM_90402]{DRAFT} ⌈An E2E-protected Event shall have its options configured in End2EndEventProtectionProps and E2EProfileConfiguration.⌋*(RS_E2E_08540)*

[SWS_CM_90433]{DRAFT} ⌈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 [7] and comply with the E2E protection protocol specification of [4] (especially [PRS_E2E_00323]).⌋*(RS_E2E_08540, RS_CM_00223)*

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.

7.2.1 Limitations

The specified E2E communication protection for events is limited.

- EndToEndTransformationComSpecProps are not supported.

General limitations regarding E2E protection and the detectable failure modes are described in [4].

The values of the following E2E parameters are defined as fixed by the standard and shall not be changed. See [PRS_E2E_00324] of [4]:

- counterOffset

- crcOffset
- dataIdNibbleOffset

The value of following E2E parameters shall be set to the default values specified by [PRS_E2E_00324] of [4]:

- offset

The value of dataIdMode for Events and the notifier of Fields 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.

7.2.2 Publisher

[SWS_CM_90453]{DRAFT} ⌈For E2E-protected Events, E2E protection shall be performed within the context of Send.⌋*(RS_CM_00223, RS_E2E_08540)*

Figure 7.5 shows an overview of the interaction of components involved during the E2E protection at the publisher side.

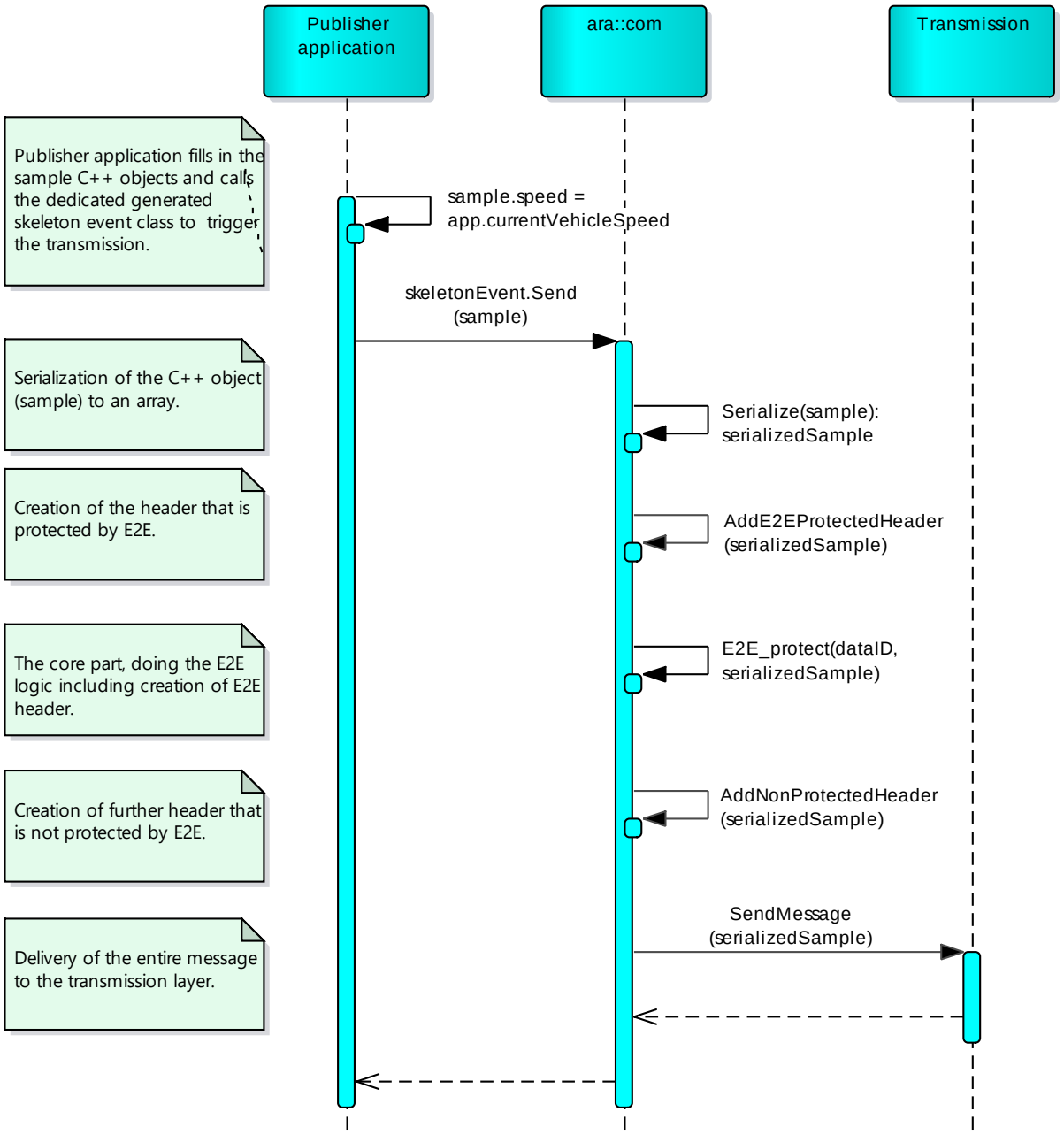


Figure 7.5: E2E Publisher

[SWS_CM_90430]{DRAFT} ⌈For E2E-protected `Events`, `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.⌋*(RS_CM_00223, RS_E2E_08540)*

From E2E protection perspective this serialized data include both a non-protected part as well as the part to be protected (see [PRS_E2E_USE_00236] and [PRS_E2E_-USE_00741]).

[SWS_CM_90401]{DRAFT} ⌈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].⌋*(RS_E2E_08540)*

[SWS_CM_90403]{DRAFT} ⌈For E2E-protected `Event`s, the `End2EndEventProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_protect`.⌋*(RS_E2E_08540)*

[SWS_CM_90404]{DRAFT} ⌈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.⌋*(RS_E2E_08540)*

7.2.3 Subscriber - GetNewSamples

[SWS_CM_90406]{DRAFT} ⌈For E2E-protected `Event`s, E2E checking shall be performed within the context of `GetNewSamples`.⌋*(RS_CM_00223, RS_E2E_08540)*

Figure 7.6 shows an overview of the interaction of components involved during the E2E checking at the subscriber side.

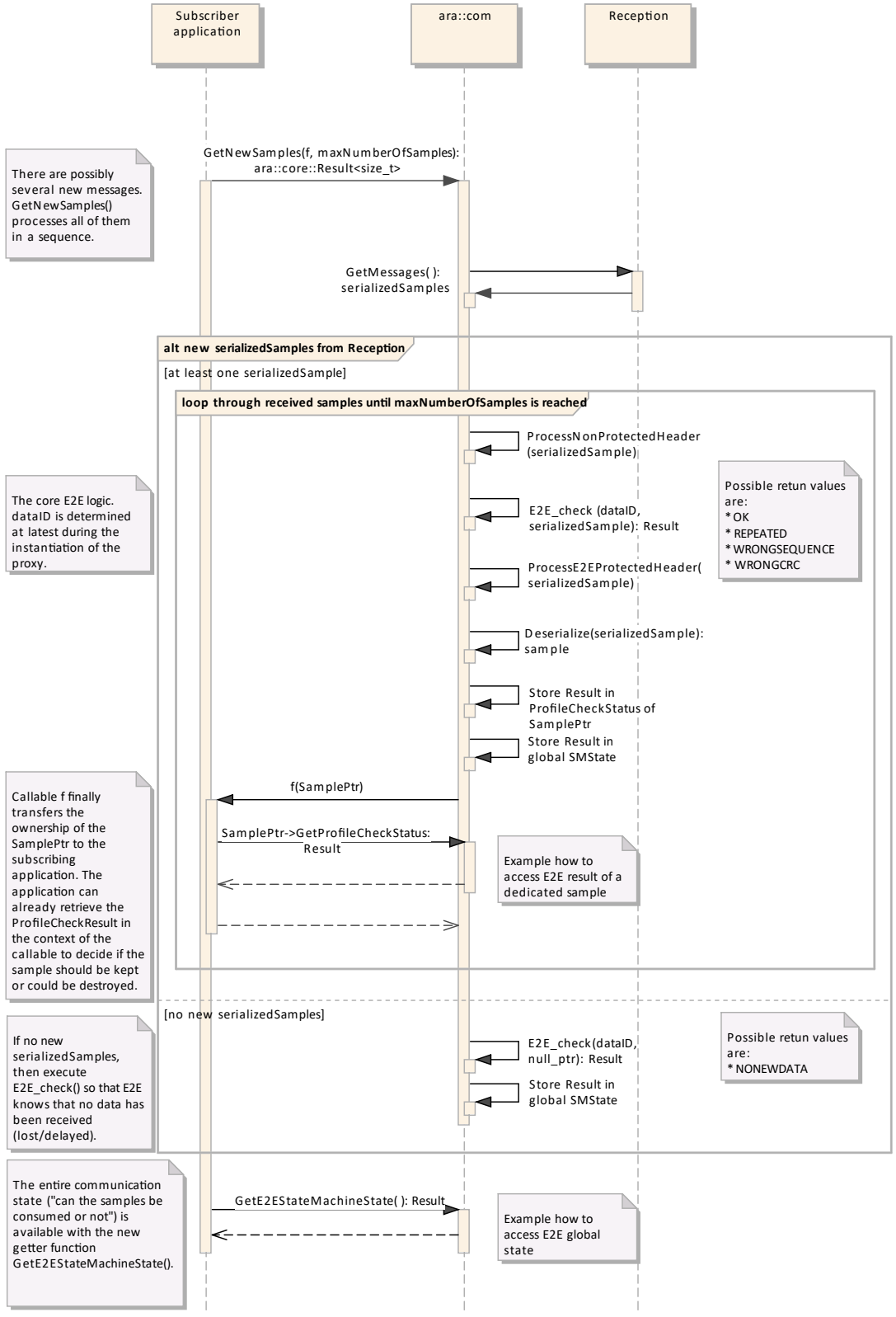


Figure 7.6: E2E Subscriber

[SWS_CM_90407]{DRAFT} ⌈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.⌋*(RS_CM_00224, RS_E2E_08540)*

From E2E protection perspective this serialized data include both a non-protected part as well as the part to be protected (see [PRS_E2E_USE_00236] and [PRS_E2E_-USE_00741]).

7.2.3.1 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]{DRAFT} ⌈For the given E2E-protected sample, `GetNewSamples` shall process the non-E2E protected header (if any) of the sample's serialized data.⌋*(RS_CM_00224, RS_E2E_08540)*

[SWS_CM_90410]{DRAFT} ⌈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].⌋*(RS_E2E_-08540)*

[SWS_CM_90454]{DRAFT} ⌈For the given E2E-protected sample, the `End2EndEventProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_check`.⌋*(RS_E2E_08540)*

[SWS_CM_90411]{DRAFT} ⌈In return, for the given E2E-protected sample, `E2E_check` shall provide a `Result` (`e2eResult` according to [PRS_E2E_00322] of [4]) containing the elements `SMState` (`e2eState` according to [PRS_E2E_00322] of [4]) and `ProfileCheckStatus` (`e2eStatus` according to [PRS_E2E_00322] of [4]).⌋*(RS_E2E_08540, RS_E2E_08534)*

[SWS_CM_90455]{DRAFT} ⌈For the given E2E-protected sample, the E2E protection header shall be removed from the serialized data.⌋*(RS_E2E_08540)*

[SWS_CM_90412]{DRAFT} ⌈For the given E2E-protected sample, `GetNewSamples` shall deserialize 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.⌋*(RS_CM_00224, RS_E2E_08540)*

[SWS_CM_90413]{DRAFT} ⌈For the given E2E-protected sample, `GetNewSamples` shall store the `ProfileCheckStatus` in the `SamplePtr` and shall update/overwrite the global `SMState` within its specific `Event` class of the specific E2E-protected `Event`.⌋*(RS_CM_00224, RS_E2E_08540, RS_E2E_08534)*

7.2.3.2 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]{DRAFT} ⌈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].⌋*(RS_E2E_08540)*

[SWS_CM_90456]{DRAFT} ⌈The End2EndEventProtectionProps.dataId shall be passed as argument dataID to E2E_check.⌋*(RS_E2E_08540)*

[SWS_CM_90416]{DRAFT} ⌈In return, for the given null sample, E2E_check shall provide a Result (e2eResult according to [PRS_E2E_00322] of [4]) containing the elements SMState (e2eState according to [PRS_E2E_00322] of [4]) and ProfileCheckStatus (e2eStatus according to [PRS_E2E_00322] of [4]).⌋*(RS_E2E_08540, RS_E2E_08534)*

[SWS_CM_90417]{DRAFT} ⌈GetNewSamples shall update/overwrite the global SMState within its specific Event class of the specific E2E-protected Event.⌋*(RS_CM_00224, RS_E2E_08540, RS_E2E_08534)*

7.2.4 Subscriber - Callable f

The user provided Callable f is invoked for each received sample. The Callable f is called with the SamplePtr of the corresponding sample as parameter. The SamplePtr contains the deserialized sample including the ProfileCheckStatus.

7.2.5 Subscriber - Access to E2E information

[SWS_CM_90457]{DRAFT} ⌈Each SamplePtr shall provide a GetProfileCheckStatus method to access the ProfileCheckStatus of each sample (see [SWS_CM_90420]).⌋*(RS_CM_00224, RS_E2E_08540)*

[SWS_CM_10475]{DRAFT} ⌈A GetE2EStateMachineState method shall be provided for each Event class of a specific ServiceProxy class.⌋*(RS_CM_00224, RS_E2E_08534)*

[SWS_CM_90431]{DRAFT} ⌈The GetE2EStateMachineState method shall provide access to the global SMState of the specific Event class, which was determined by the last run of E2E_check function invoked during the last call of GetNewSamples (see [SWS_CM_90417]).⌋*(RS_CM_00224, RS_E2E_08534)*

```
ara::com::e2e::SMState GetE2EStateMachineState() const noexcept;
```

7.3 End-to-end communication protection for 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]{DRAFT} ⌈An E2E-protected `Method` shall have its options configured in `End2EndMethodProtectionProps` and `E2EProfileConfiguration`.⌋ *(RS_CM_00400, RS_E2E_08540)*

[SWS_CM_90485]{DRAFT} ⌈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 [7] and comply with the E2E protection protocol specification of [4] (especially [PRS_E2E_00828]).⌋ *(RS_CM_00400, RS_E2E_08541, RS_CM_00223)*

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`, `NOTAVAILABLE`, or `NONEWDATA`. A successful E2E check is an invocation of `E2E_check` returning an `e2eStatus` different from `REPEATED`, `WRONGSEQUENCE`, `NOTAVAILABLE`, and `NONEWDATA`.

7.3.1 Limitations

The specified E2E communication protection for `method`s is limited.

- The processing mode 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 [4].

The values of the following E2E parameters are defined as fixed by the standard and shall not be changed. See [PRS_E2E_00324] of [4]:

- `counterOffset`
- `crcOffset`
- `dataIdNibbleOffset`

The value of following E2E parameters shall be set to the default values specified by [PRS_E2E_00324] of [4]:

- offset

The value of dataIdMode for Methods and the getters and setters of Fields 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.

7.3.2 E2E protection of the service method request (Client)

[SWS_CM_10462]{DRAFT} ⌈For E2E-protected Methods, 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.⌋*(RS_CM_00400, RS_E2E_08541)*

Figure 7.7 shows an overview of the interaction of components involved during the E2E protection of the Method request at the client side.

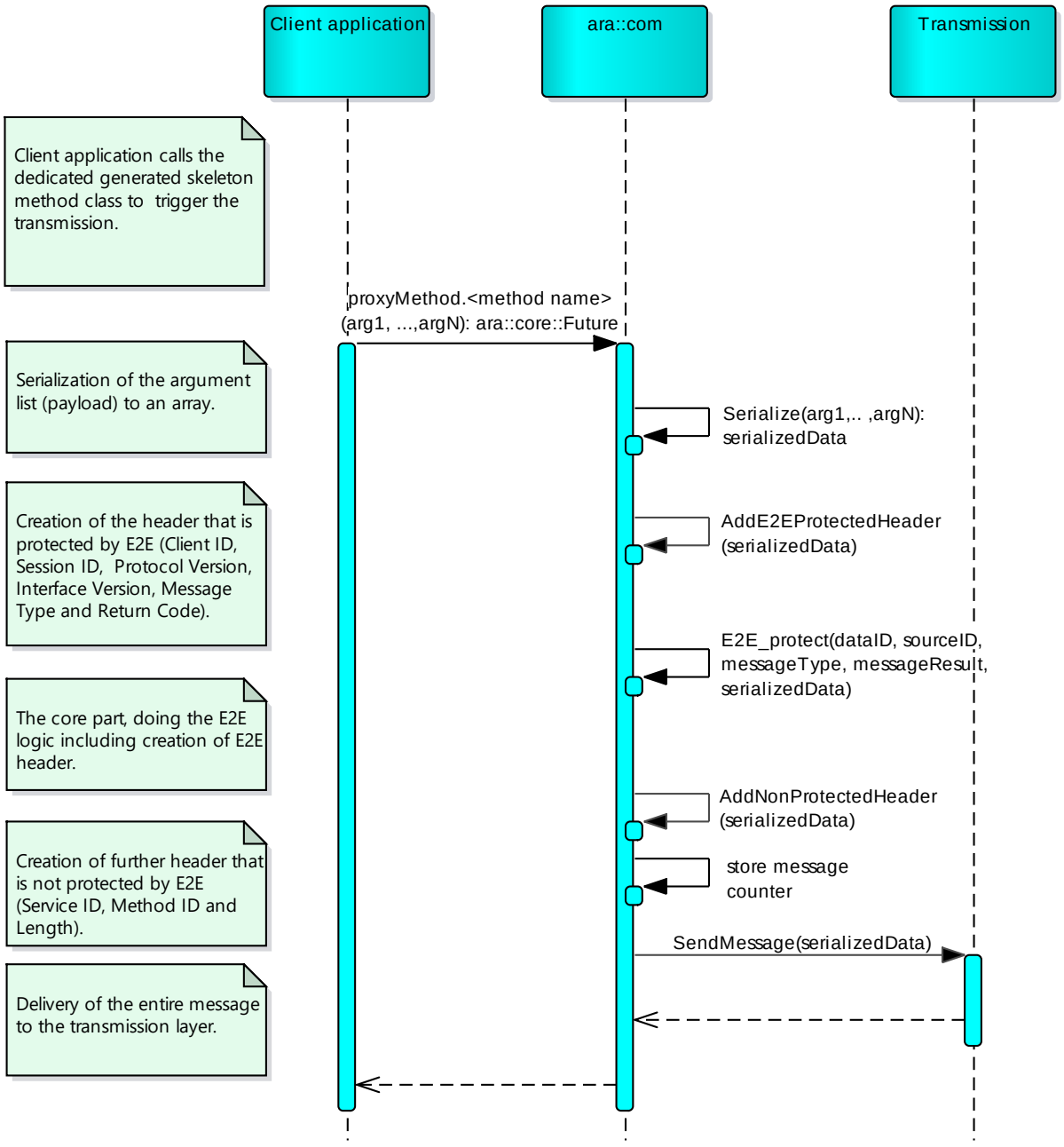


Figure 7.7: Interaction of components during E2E protection of the `Method` request at the client side

7.3.2.1 Serializing the payload

[SWS_CM_90458]{DRAFT} ⌈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.⌋*(RS_CM_00400, RS_E2E_08541)*

From E2E protection perspective this serialized data include both a non-protected part as well as the part to be protected (see [PRS_E2E_USE_00236] and [PRS_E2E_-USE_00741]).

7.3.2.2 E2E protection of the payload

[SWS_CM_90479]{DRAFT} ⌈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].⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_10463]{DRAFT} ⌈For E2E-protected `Method` requests, the `End2EndMethodProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_protect`.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90486]{DRAFT} ⌈For E2E-protected `Method` requests using profiles P04m, P07m, P08m, or P44m, the `End2EndMethodProtectionProps.sourceId` shall be passed as argument `sourceID` to `E2E_protect`.⌋*(RS_CM_00400, RS_-E2E_08541)*

[SWS_CM_90487]{DRAFT} ⌈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`.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90488]{DRAFT} ⌈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`.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_10464]{DRAFT} ⌈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.⌋*(RS_CM_00400, RS_E2E_08541)*

7.3.3 E2E checking the service method request (Server)

[SWS_CM_10466]{DRAFT} ⌈For E2E-protected `Method` requests, E2E checking shall be performed within the context of the message reception within the `ServiceSkeleton` if the `MethodCallProcessingMode` is set to `kEventSingleThread`.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_10468]{DRAFT} ⌈For E2E-protected `Method` requests, E2E checking shall be performed within the context of `ProcessNextMethodCall` within the `Ser-`

viceSkeleton if the MethodCallProcessingMode is set to kPoll.⌋(*RS_CM_00400, RS_E2E_08541*)

[SWS_CM_10467]{DRAFT} ⌈In case a MethodCallProcessingMode of kEvent has been passed to the named constructor of the ServiceSkeleton for a service using e2e-protected methods (see [SWS_CM_10436] or [SWS_CM_10435]), an error code kWrongMethodCallProcessingMode shall be returned in the Result of the named constructor Create(). If logging is enabled, the error shall be logged.⌋(*RS_CM_00402, RS_CM_00400, RS_E2E_08541*)

Note: A MethodCallProcessingMode set to kEvent is not supported for E2E-protected Methods.

Figures 7.8 and 7.9 show an overview of the interaction of components involved during the E2E checking of the Method request at the server side.

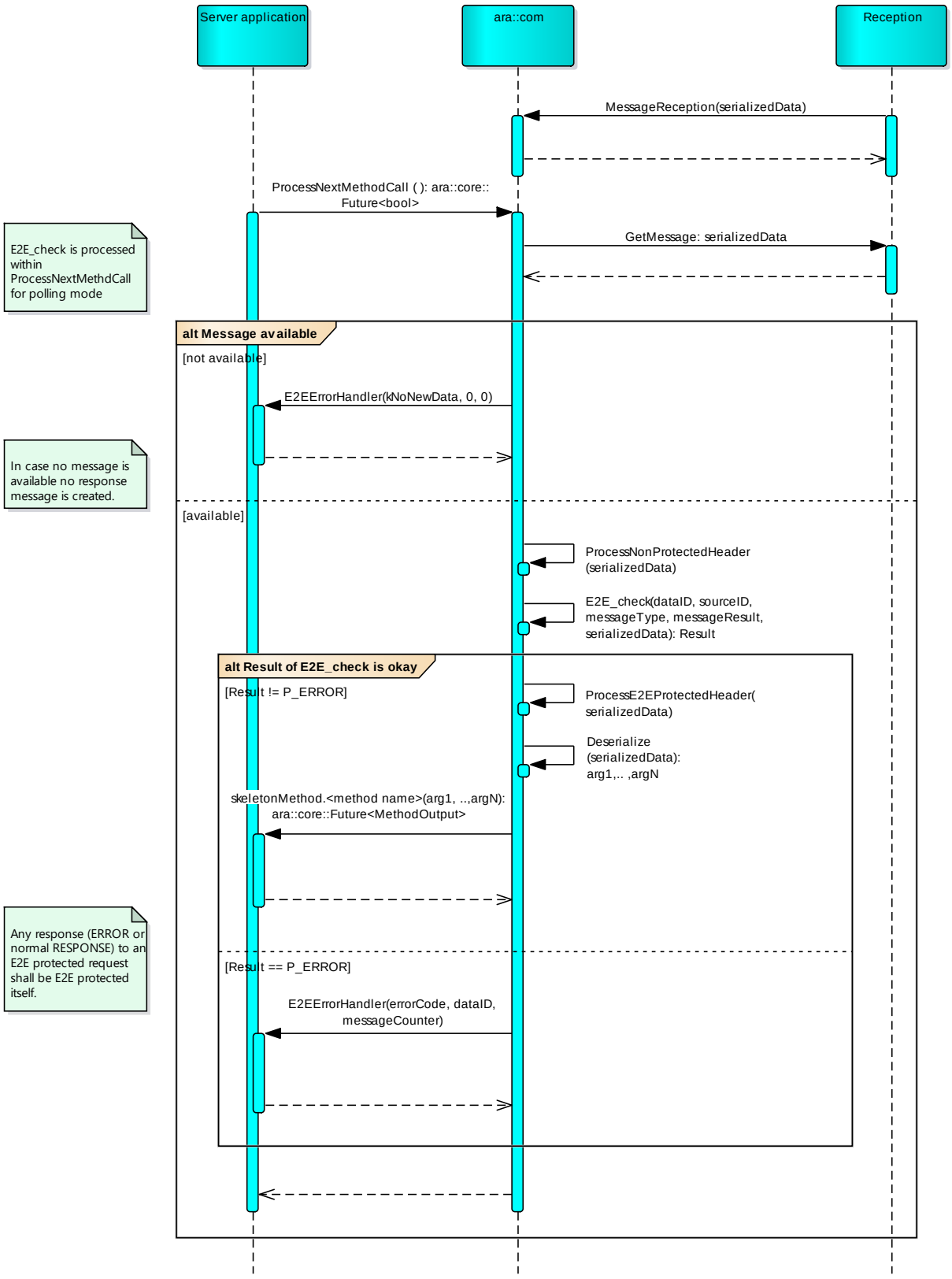


Figure 7.8: Interaction of components during E2E checking of the `Method` request at the server side - polling

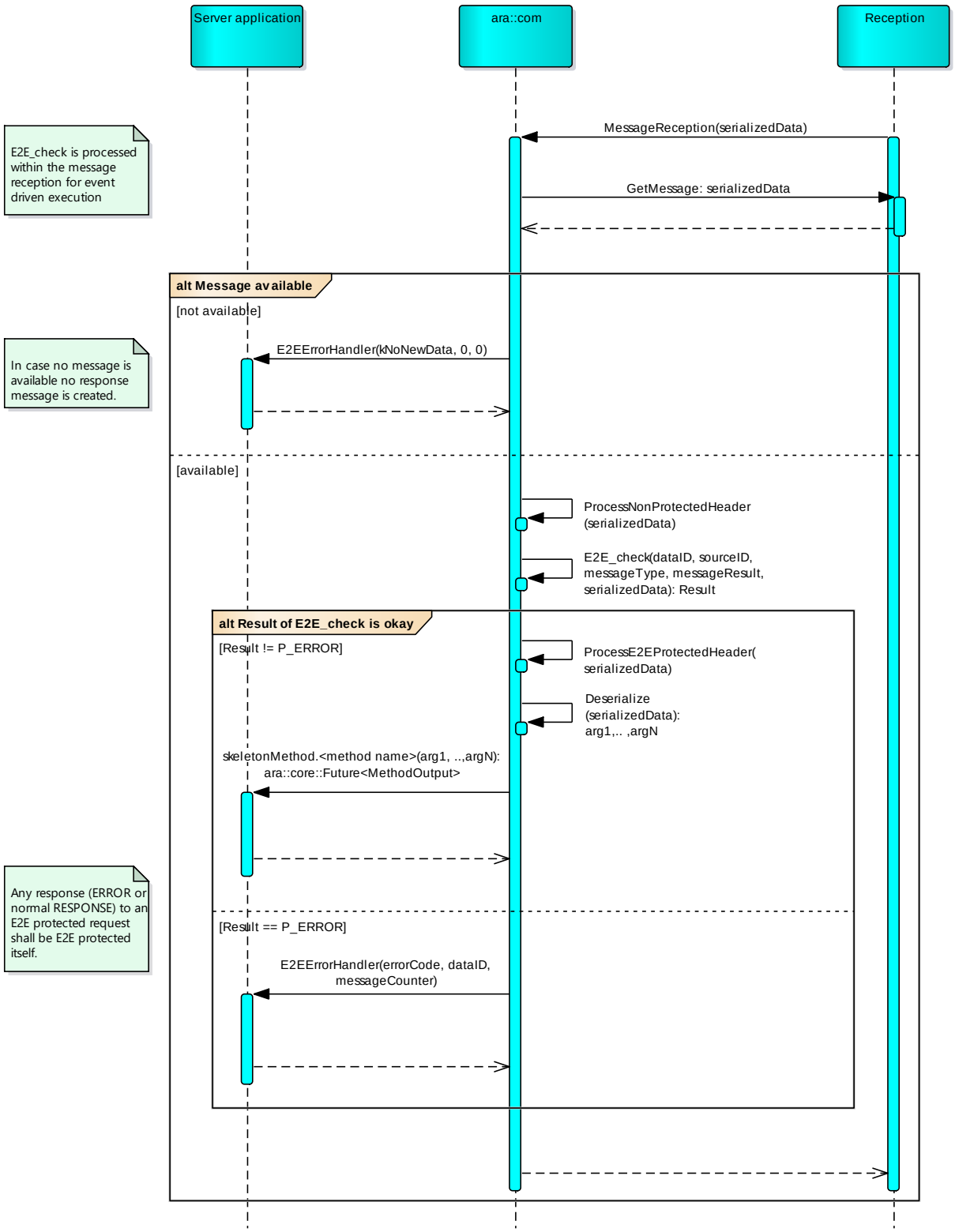


Figure 7.9: Interaction of components during E2E checking of the `Method` request at the server side - event driven

7.3.3.1 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_90459]{DRAFT} ⌈For the given E2E-protected Method request, the non-E2E-protected header (if any) of the Method request's serialized data shall be processed.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90480]{DRAFT} ⌈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].⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90460]{DRAFT} ⌈For the given E2E-protected Method request, the End2EndMethodProtectionProps.dataId shall be passed as argument dataID to E2E_check()).⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90489]{DRAFT} ⌈For E2E-protected Method requests using profiles P04m, P07m, P08m, or P44m, a reference to a variable to store the 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]).⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90490]{DRAFT} ⌈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.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90491]{DRAFT} ⌈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.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90461]{DRAFT} ⌈In return, for the given E2E-protected Method request, E2E_check shall provide a Result (e2eResult according to [PRS_E2E_00322] of [4]) containing the elements SMState (e2eState according to [PRS_E2E_00322] of [4]) and ProfileCheckStatus (e2eStatus according to [PRS_E2E_00322] of [4]).⌋*(RS_E2E_08541, RS_E2E_08534)*

[SWS_CM_90462]{DRAFT} ⌈For the given E2E-protected Method request, the E2E protection header shall be removed from the serialized data.⌋*(RS_CM_00400, RS_E2E_08541)*

7.3.3.2 Deserializing the payload

In case the call to E2E_check (according to [SWS_CM_90459]) indicated a successful E2E check of the request message further processing of the request message shall take place.

[SWS_CM_90463]{DRAFT} ⌈For the given E2E-protected `Method` request, the resulting serialized data shall be deserialized 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 deserialized `in` and `inout` arguments to the `Method` call.⌋ *(RS_CM_00400, RS_E2E_08541)*

7.3.3.3 E2E error notification

In case the call to `E2E_check` (according to [SWS_CM_90459]) indicated a failed E2E check of the request message, the server application can get notified via an E2E error handler.

[SWS_CM_10470]{DRAFT} **E2E Error Handler - Existence** ⌈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::E2EErrorCode errorCode,
    ara::com::e2e::DataID dataID,
    ara::com::e2e::MessageCounter messageCounter
    )
{
};
```

⌋*(RS_CM_00401, RS_CM_00402)*

[SWS_CM_90464]{DRAFT} **E2E Error Handler - Invocation** ⌈`E2EErrorHandler` shall be invoked from within a separate thread by the Communication Management software in case `E2E_check` reports an E2E error.⌋*(RS_CM_00401, RS_CM_00402)*

[SWS_CM_10471]{DRAFT} **E2E Error Handler - Invocation Arguments** ⌈In case a new request message is available, `E2EErrorHandler` shall be called with the following arguments: `errorCode` shall be set to the `ProfileCheckStatus` obtained in [SWS_CM_90411], `dataID` shall be set to `End2EndMethodProtectionProps.dataId`, and `messageCounter` shall be set to the E2E counter of the received request message.⌋*(RS_CM_00223, RS_CM_00401, RS_CM_00402)*

[SWS_CM_90465]{DRAFT} **E2E Error Handler - Invocation Arguments** ⌈In case no new request message is available, `E2EErrorHandler` shall be called with the following arguments: `errorCode` shall be set to the `kNotAvailable`, `dataID` shall be set to `0`, and `messageCounter` shall be set `0`.⌋*(RS_CM_00401, RS_CM_00402)*

7.3.4 E2E protection of the service method response (Server))

[SWS_CM_90481]{DRAFT} ⌈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]).⌋ *(RS_CM_00400, RS_E2E_08541)*

Figure 7.10 shows an overview of the interaction of components involved during the E2E protection of the `Method` response at the server side.

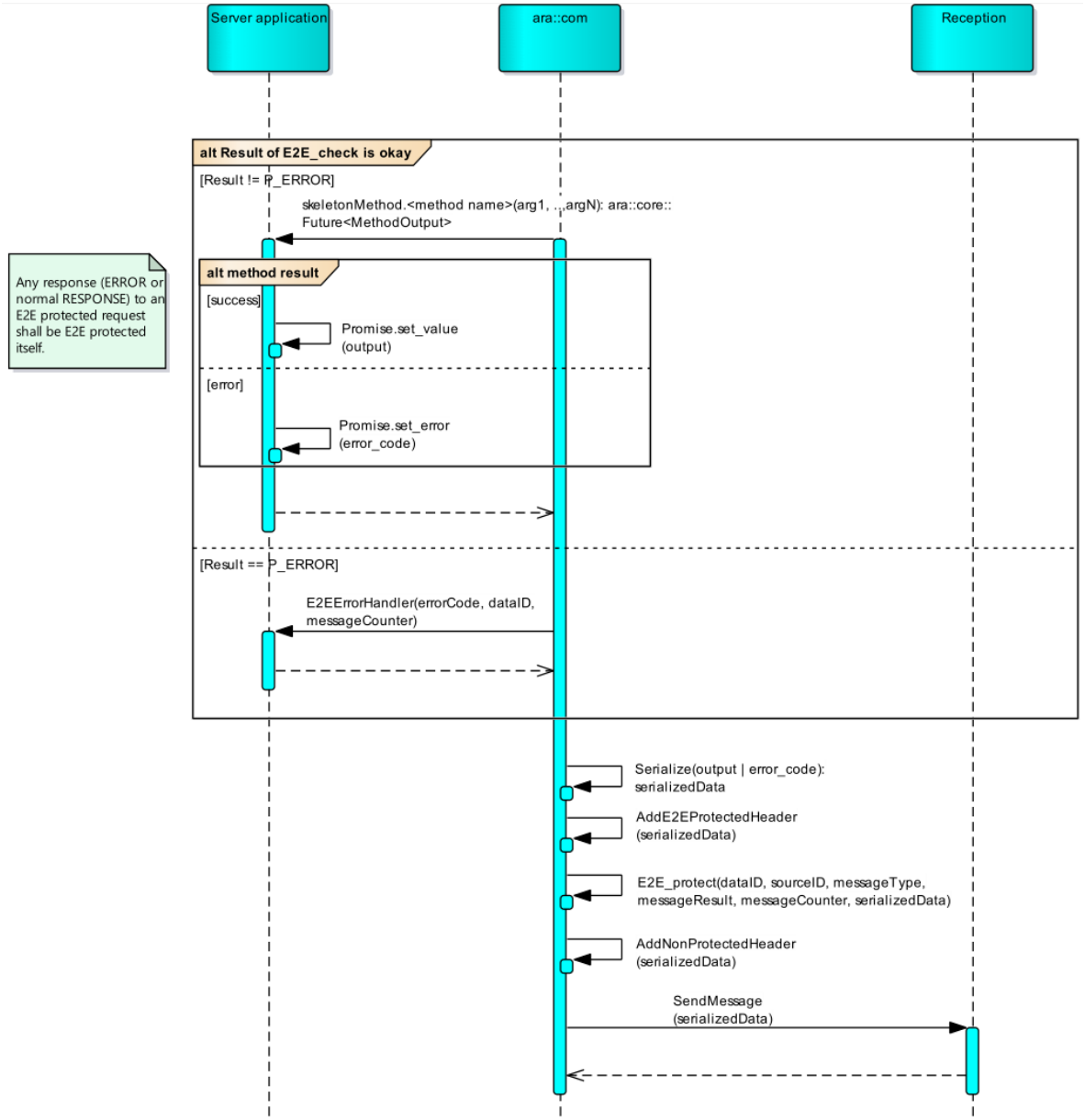


Figure 7.10: Interaction of components during E2E protection of the `Method` response at the server side

7.3.4.1 Serializing the E2E error response payload

[SWS_CM_10472]{DRAFT} **E2E Error Response** ⌈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.⌋*(RS_CM_00223, RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90466]{DRAFT} **Payload of the E2E Error Response** ⌈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).⌋*(RS_CM_00400, RS_E2E_08541)*

7.3.4.2 Serializing the response payload

[SWS_CM_90467]{DRAFT} **Payload of the Normal or Application Error Response** ⌈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.⌋*(RS_CM_00400, RS_E2E_08541)*

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_USE_00236] and [PRS_E2E_USE_00741]).

7.3.4.3 E2E protection of the response payload

[SWS_CM_90468]{DRAFT} ⌈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].⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_10469]{DRAFT} ⌈For E2E-protected `Method` responses, the `End2EndMethodProtectionProps.dataId` shall be passed as argument `dataID` to `E2E_protect`.⌋*(RS_CM_00400, RS_E2E_08541)*

Note: This is the same `dataID` that has been contained in the corresponding `Method` request.

[SWS_CM_90492]{DRAFT} ⌈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 shall be passed as argument `sourceID` to `E2E_protect`.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90493]{DRAFT} ⌈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`.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90494]{DRAFT} ⌈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`.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90495]{DRAFT} ⌈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`.⌋*(RS_CM_00401, RS_E2E_08541)*

[SWS_CM_90469]{DRAFT} ⌈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`.⌋*(RS_CM_00400, RS_E2E_08541)*

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]{DRAFT} ⌈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.⌋*(RS_CM_00400, RS_E2E_08541)*

7.3.5 E2E checking the service method response (Client)

[SWS_CM_90471]{DRAFT} ⌈For E2E-protected `Method` responses, E2E checking shall be performed within the context of the message reception within the `ServiceProxy`.⌋*(RS_CM_00400, RS_E2E_08541)*

Figure 7.11 shows an overview of the interaction of components involved during the E2E checking of the `Method` response at the client side.

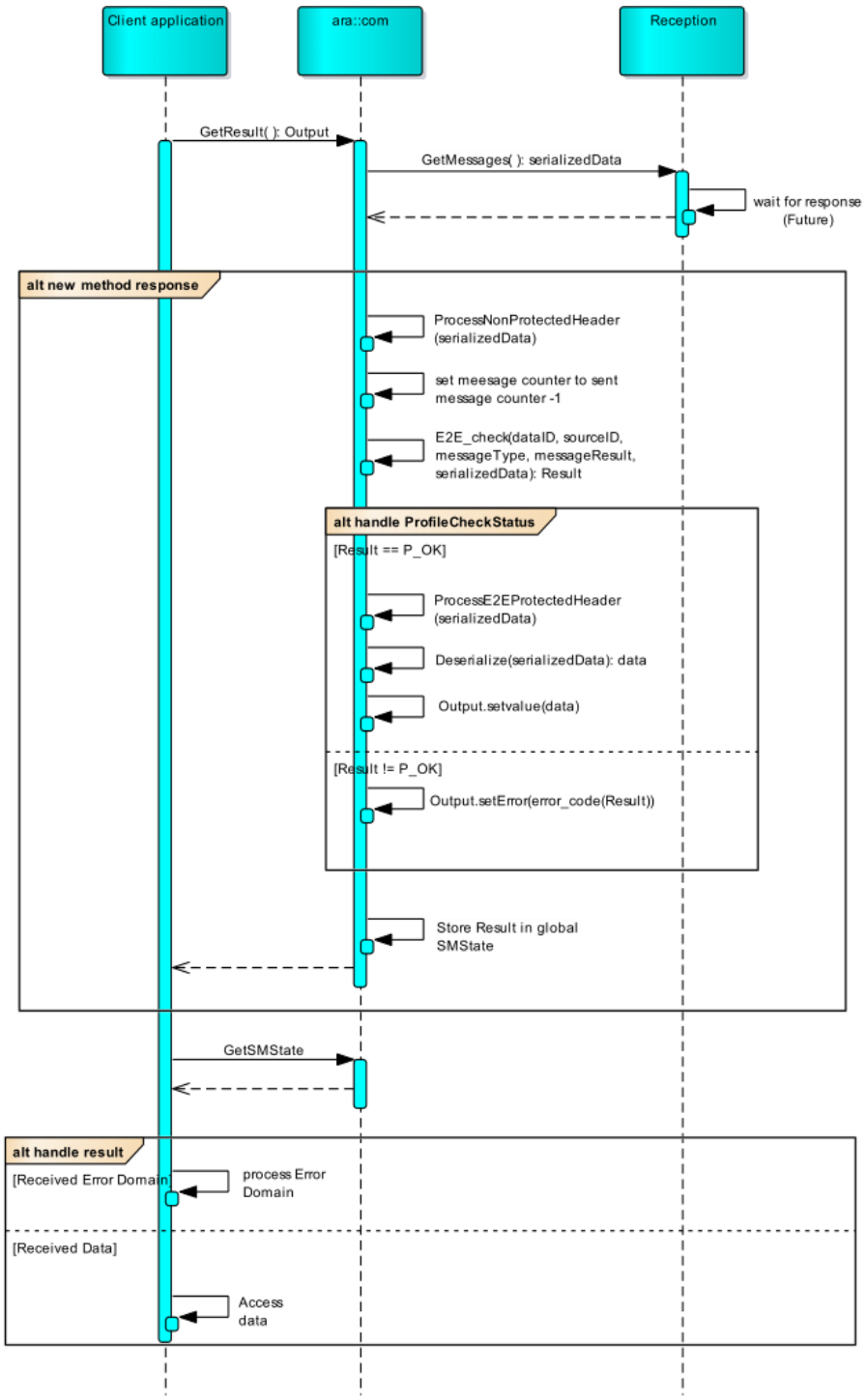


Figure 7.11: Interaction of components during E2E checking of the `Method` response at the client side

7.3.5.1 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]{DRAFT} ⌈For the given E2E-protected Method responses, the non-E2E-protected header (if any) of the Method response's serialized data shall be processed.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90473]{DRAFT} ⌈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].⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90474]{DRAFT} ⌈For the given E2E-protected Method response, the End2EndMethodProtectionProps.dataId shall be passed as argument dataID to E2E_check()).⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_10465]{DRAFT} ⌈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).⌋*(RS_CM_00400, RS_E2E_08541)*

Implementation Hint: The E2E counter can be extracted from the resulting state of the E2E_Protect()/E2E_Check() function.

[SWS_CM_90496]{DRAFT} ⌈For E2E-protected Method responses using profiles P04m, P07m, P08m, or P44m, the End2EndMethodProtectionProps.sourceId shall be passed as argument sourceID to E2E_check.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90497]{DRAFT} ⌈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.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90498]{DRAFT} ⌈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.⌋*(RS_CM_00400, RS_E2E_08541)*

[SWS_CM_90499]{DRAFT} ⌈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.⌋*(RS_CM_00401, RS_E2E_08541)*

[SWS_CM_90478]{DRAFT} ⌈In return, for the given E2E-protected Method response, E2E_check shall provide a Result (e2eResult according to [PRS_E2E_00322] of [4]) containing the elements SMState (e2eState according to [PRS_E2E_00322] of

[4]) and ProfileCheckStatus (e2eStatus according to [PRS_E2E_00322] of [4]).⌋ *(RS_E2E_08541, RS_E2E_08534)*

[SWS_CM_90482]{DRAFT} ⌈The global SMState within its specific Method class of a specific ServiceProxy class shall be updated/overwriten with the element SM-State of the Result provided by E2E_check according to [SWS_CM_90478].⌋*(RS_-CM_00400, RS_E2E_08541, RS_E2E_08534)*

[SWS_CM_90475]{DRAFT} ⌈For the given E2E-protected Method response, the E2E protection header shall be removed from the serialized data.⌋*(RS_CM_00400, RS_-E2E_08541)*

7.3.5.2 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]{DRAFT} ⌈For the given E2E-protected Method response, the resulting serialized data shall be deserialized 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 deserialized application error.⌋*(RS_CM_00400, RS_E2E_-08541)*

[SWS_CM_10473]{DRAFT} **Handling the E2E Error Response** ⌈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).⌋*(RS_CM_00223, RS_CM_00400, RS_E2E_08541)*

7.3.5.3 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]{DRAFT} **E2E Error Return Code** ⌈For the given E2E-protected Method response in case of failed E2E check an ara::core::ErrorCode of error domain ara::com::e2e::E2EErrorDomain with value set to Pro-fileCheckStatus obtained in [SWS_CM_90478] shall be constructed according to [SWS_CM_90421]. This ara::core::ErrorCode shall be passed as argument in a call to SetError() on the Promise.⌋*(RS_CM_00400, RS_E2E_08541)*

The handling of normal and application error responses (according to [SWS_CM_90476]) combined with the handlling 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::E2EErrorDomain` and the `value` set to the `ProfileCheckStatus` 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::E2EErrorDomain` and the `value` set to the `ProfileCheckStatus` 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]{DRAFT} ⌈A `GetE2EStateMachineState` method shall be provided for each `Method` class of a specific `ServiceProxy` class.⌋*(RS_E2E_08534)*

[SWS_CM_90484]{DRAFT} ⌈The `GetE2EStateMachineState` method shall provide access to the global `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]).⌋*(RS_E2E_08534)*

```
ara::com::e2e::SMState GetE2EStateMachineState() const noexcept;
```

7.3.6 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 End-to-end communication protection for Fields

This section specifies E2E protection for fields. For details of fields see [5]. 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 7.3). 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 7.3.1 are applicable.

The E2E protection for Update follows the E2E protection for events (chapter 7.2). The specifications [SWS_CM_90402] and [SWS_CM_90433] define the parameters for E2E protection of the update event. The limitations of chapter 7.2.1 are applicable.

E2E results OK and OK_SOME_LOST are successful results. E2E results ERROR, REPEATED, WRONGSEQUENCE, NOTAVAILABLE 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 Send a GET message

The client application calls the Get() function at ara::com without arguments. A 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_90458] 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 7.3.2.2.

After E2E protection the non E2E protected part is added to the message as described in [SWS_CM_10464].

Figure 7.12 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.

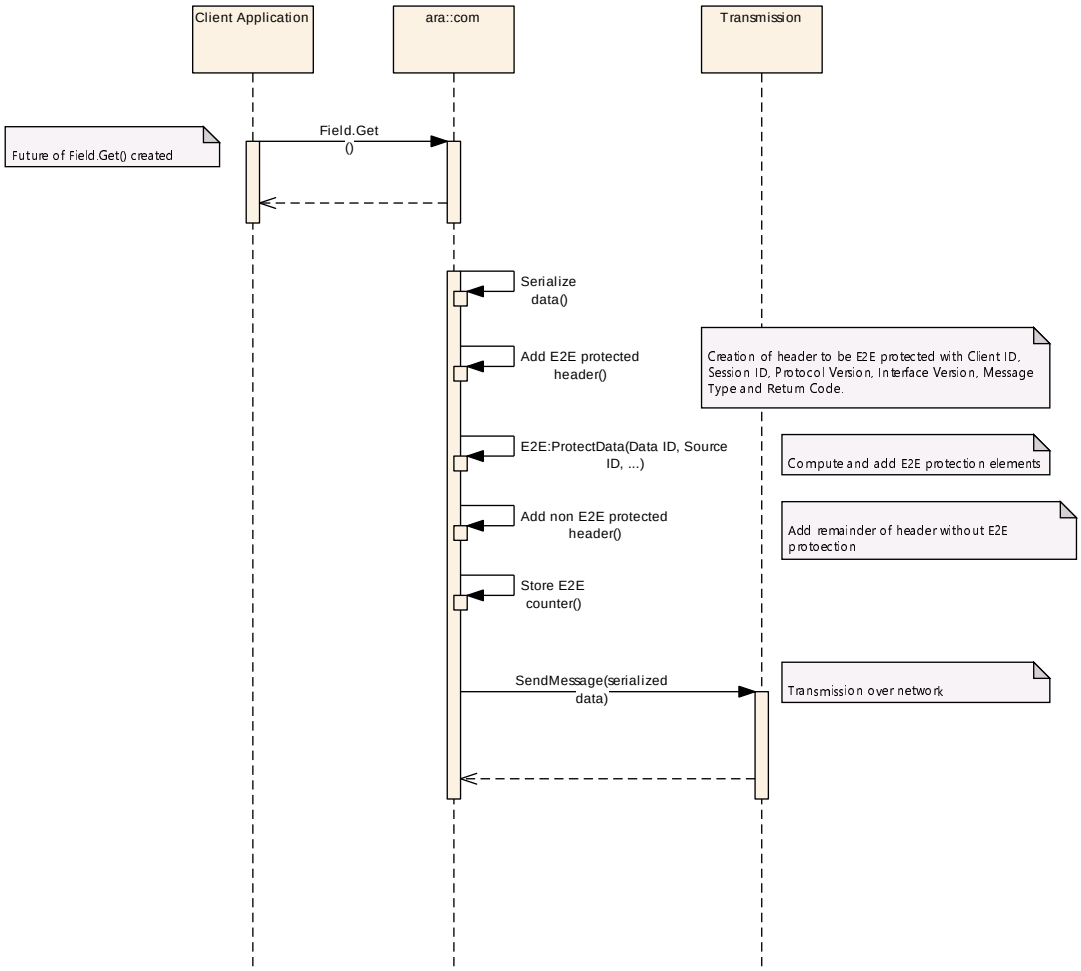


Figure 7.12: Send a GET Message

7.4.2 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 7.3.3.

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 7.13 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 7.3.3.3). 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 7.3.4.1, 7.3.4.2 and 7.3.4.3.

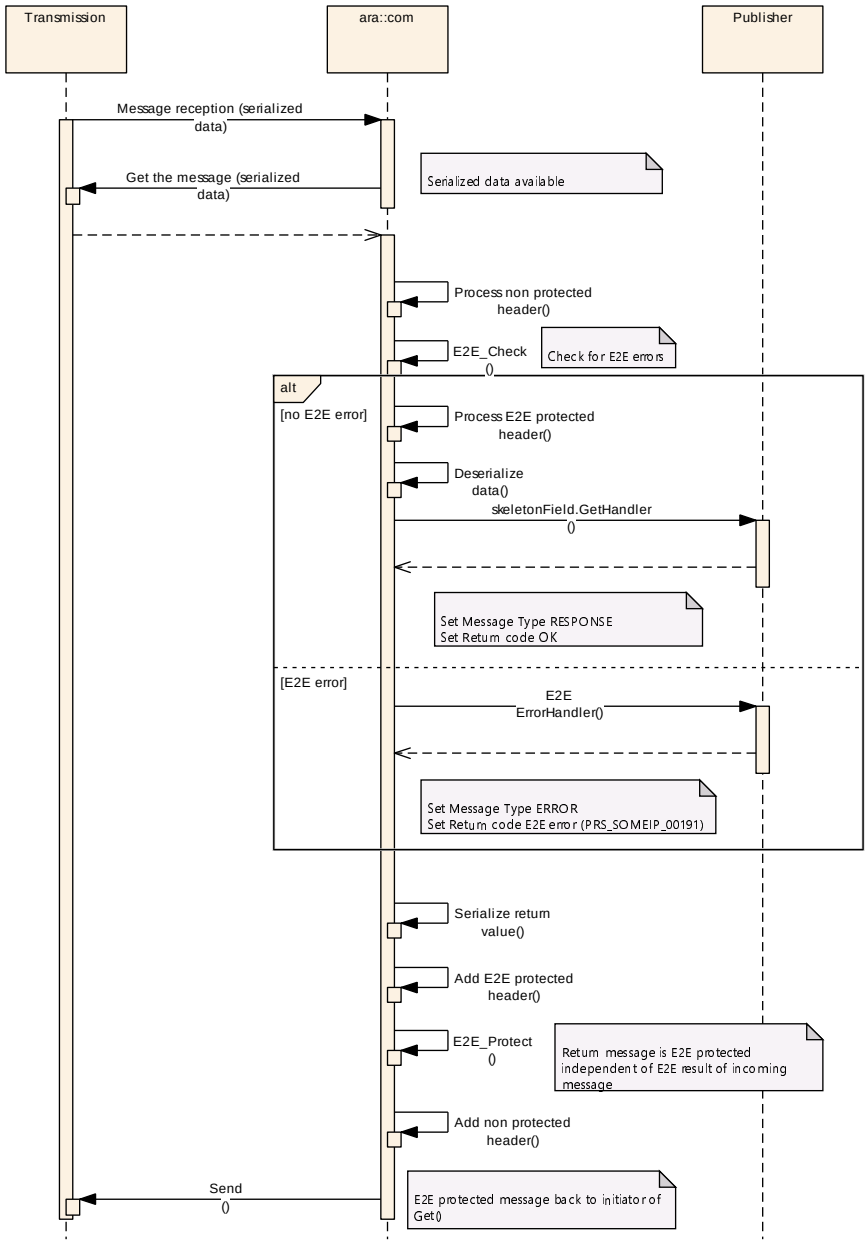


Figure 7.13: Receive a GET Message

7.4.3 Receive a response to a GET message

The reception of an E2E protected response message is described in chapter 7.3.5.

If the message is received with an E2E error, then the E2E Errorhandler of the client is called. The `future` of the `Get()` function is set to ready state with an error code. That is described in chapter 7.3.5.

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 `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 `future` is updated with the received value of the Publishers field. The `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 `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 7.3.6) may unlock the `future`. Figure 7.14 shows reception of a message from the server.

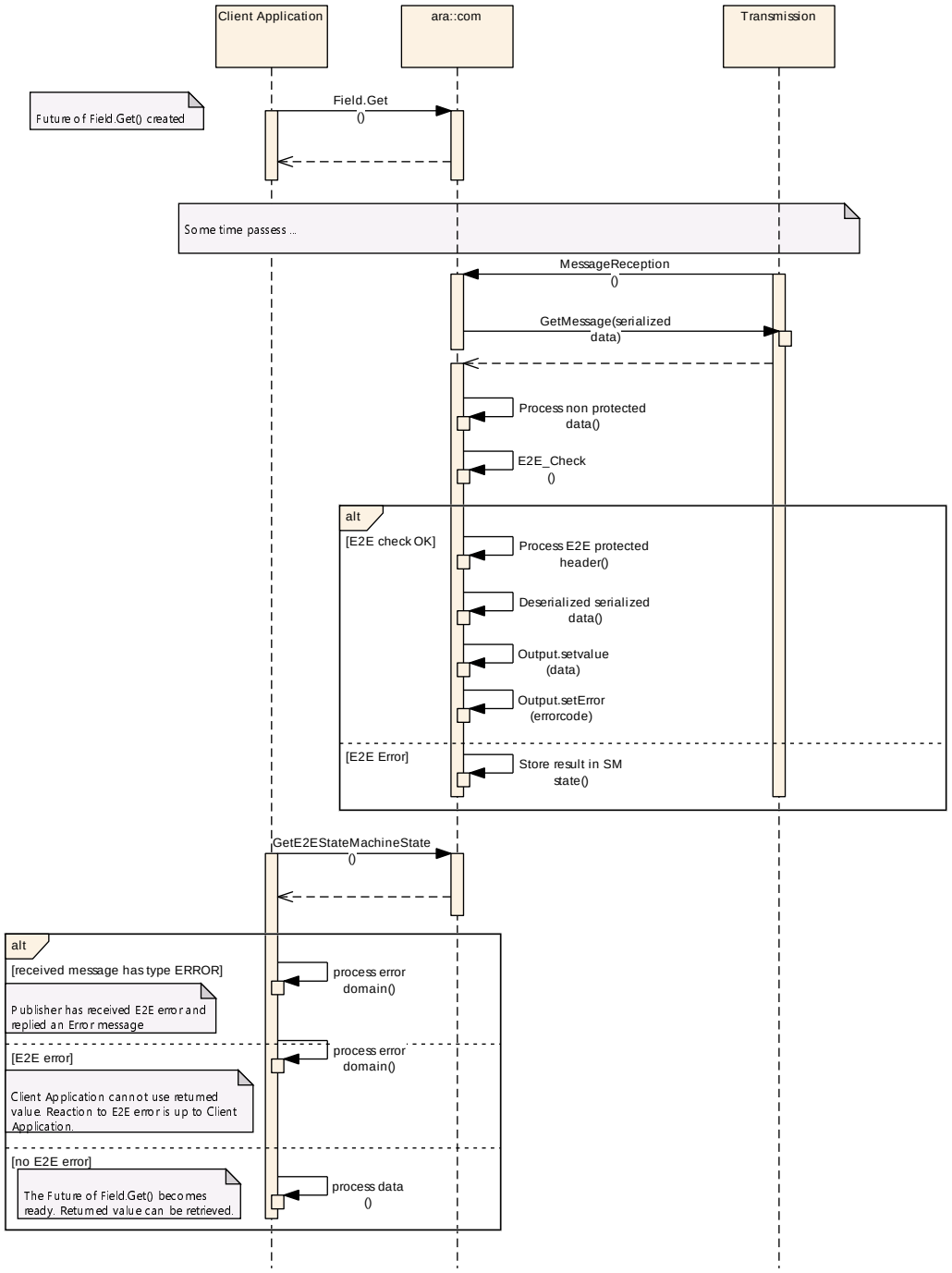


Figure 7.14: Receive response to a GET Message

7.4.4 Send a SET message

The E2E serialization follows the specification of [SWS_CM_90458]. 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 7.3.2.2.

After E2E protection the non E2E protected part is added to message as described in specification [SWS_CM_10464].

Figure 7.15 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).

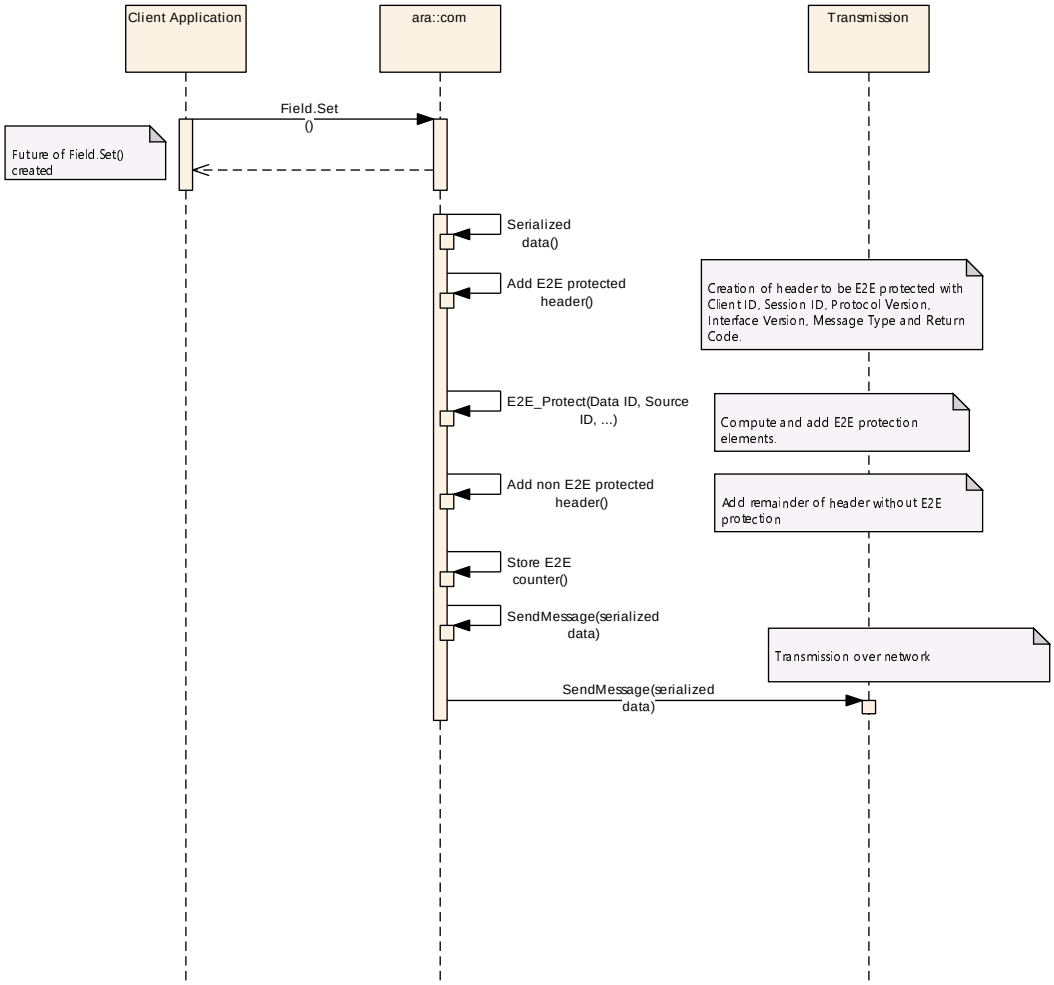


Figure 7.15: Send a SET Message

7.4.5 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 7.3.3.

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 7.3.3.3). 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 7.3.4.1, 7.3.4.2 and 7.3.4.3.

Figure 7.16 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.

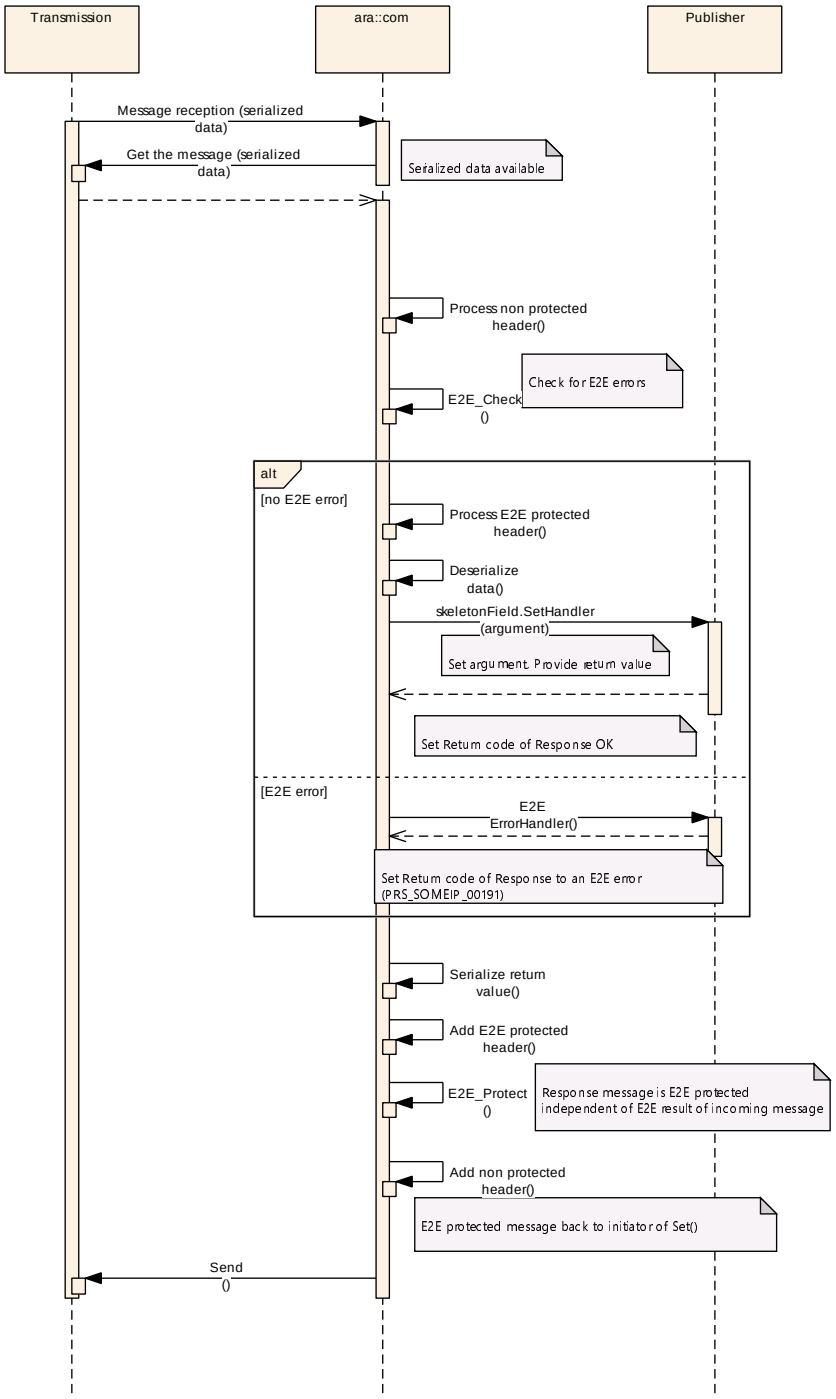


Figure 7.16: Receive a SET Message

7.4.6 Receive a response to a SET message

The reception of an E2E protected response message is described in chapter 7.3.5.

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 7.3.5.3.

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 `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 `future` is updated with the received value of Publisher's field. The `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 `future` of `Field.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 7.3.6) can unlock the `future`.

Figure 7.17 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.

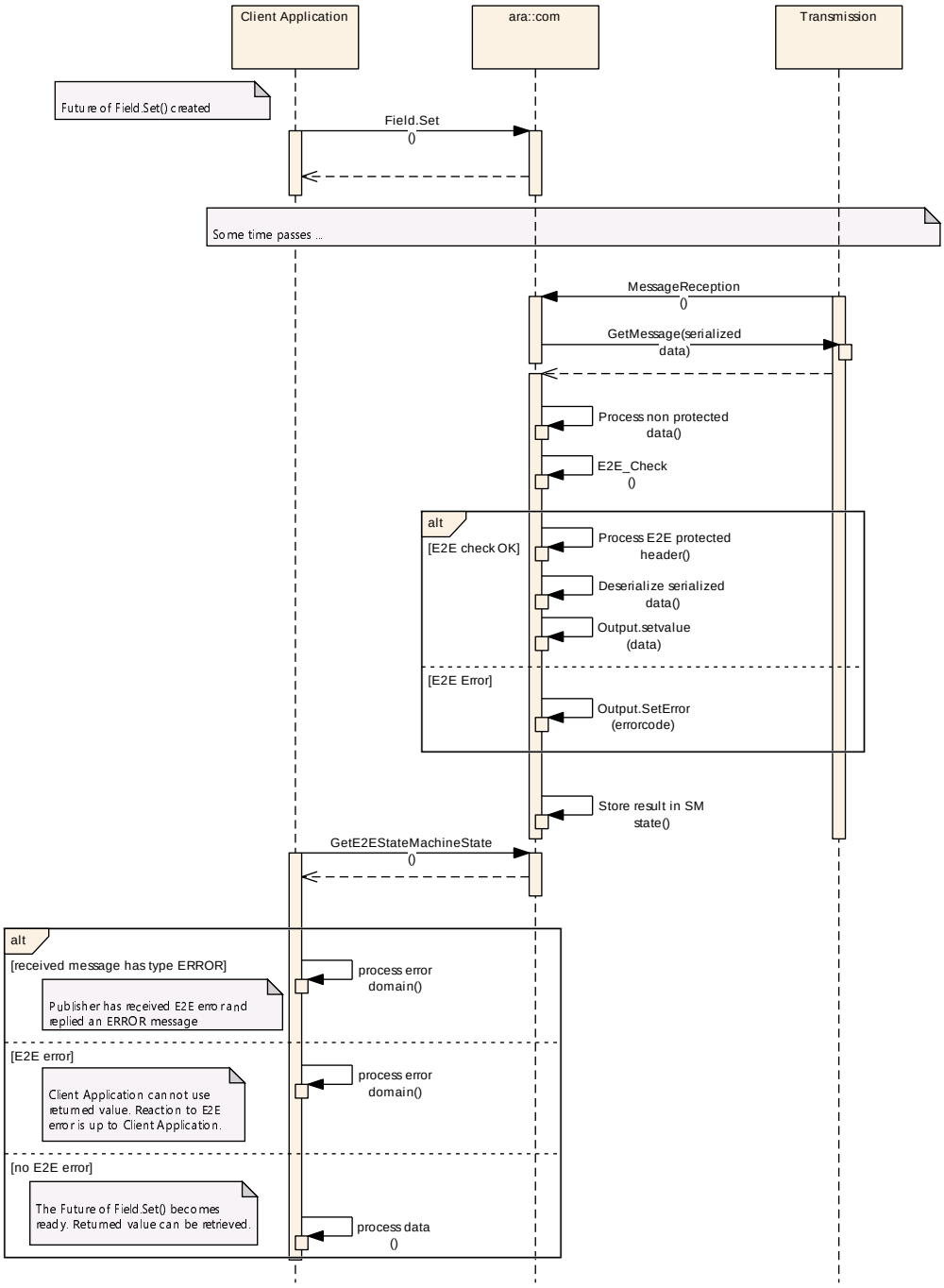


Figure 7.17: Receive response to a SET Message

7.4.7 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 7.2.2. The update message is sent to every subscriber to the publisher's field.

Figure 7.18 shows sending of field update messages.

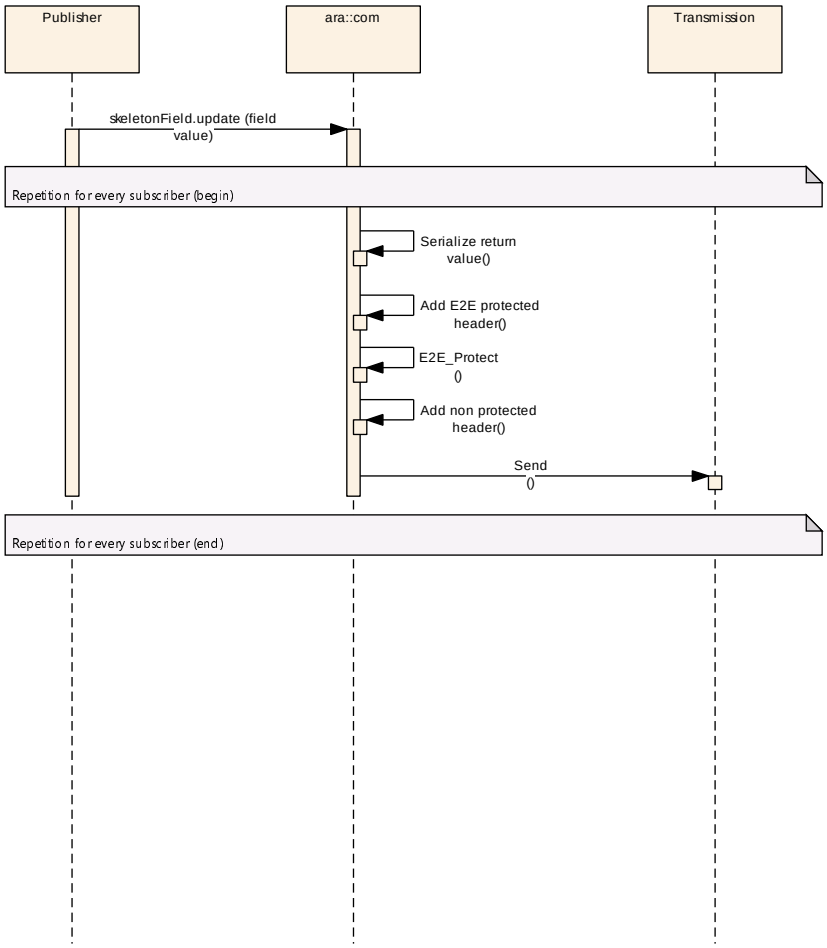


Figure 7.18: Send an UPDATE Message

7.4.8 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 7.2.3.

The reception of E2E protected fields follows the principle of E2E protected events (see figure 7.6 in chapter 7.2). 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 7.19 shows reception of a field update message.

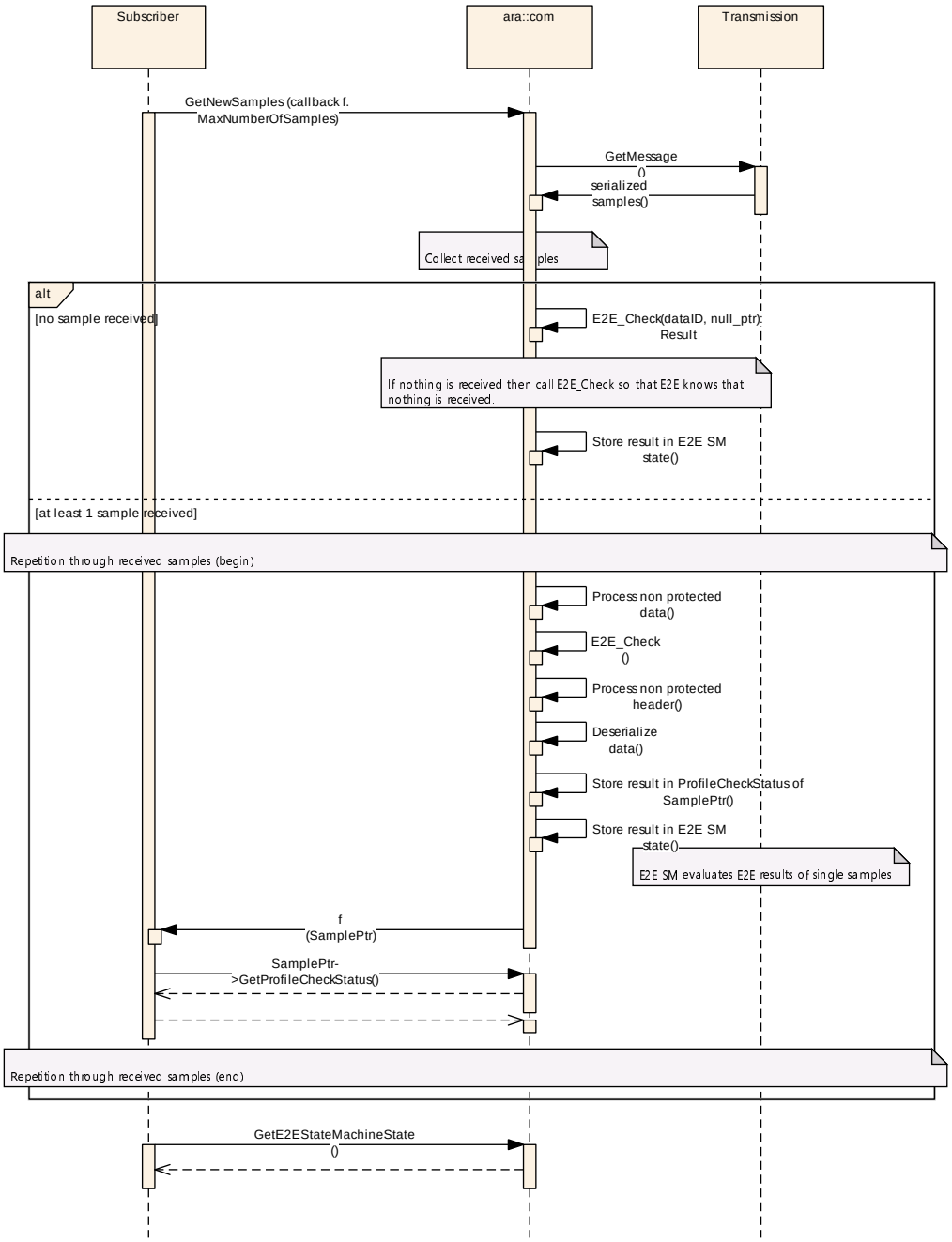


Figure 7.19: Receive an UPDATE Message

7.5 Raw Data Streaming

7.5.1 Raw Data Streaming Interface

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

of the data is not necessary. This section specifies an interface as part of `ara::com` to support processing of raw binary data streams, as an alternative to SOME/IP.

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

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

The integration of the Raw Data Streaming Interface and Adaptive Applications is done in the deployment phase, by specifying various attributes and parameters for the socket connections that shall be used for the Raw Data Stream, using `RawDataStreamMapping` and `EthernetRawDataStreamMapping`. The model and the parameters are described in *TPS_ManifestSpecification* [6].

Secure communication can be achieved by applying TLS or IPSec protocols in the middleware. Also access control imposed by the IAM can be applied for Raw Data Streams. All security functions are configurable in the deployment and mapping model of Raw Data Streaming Interface, see *TPS_ManifestSpecification* [6].

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

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

Figure 7.20 shows the logical view of the usage of RawDataStream instances.

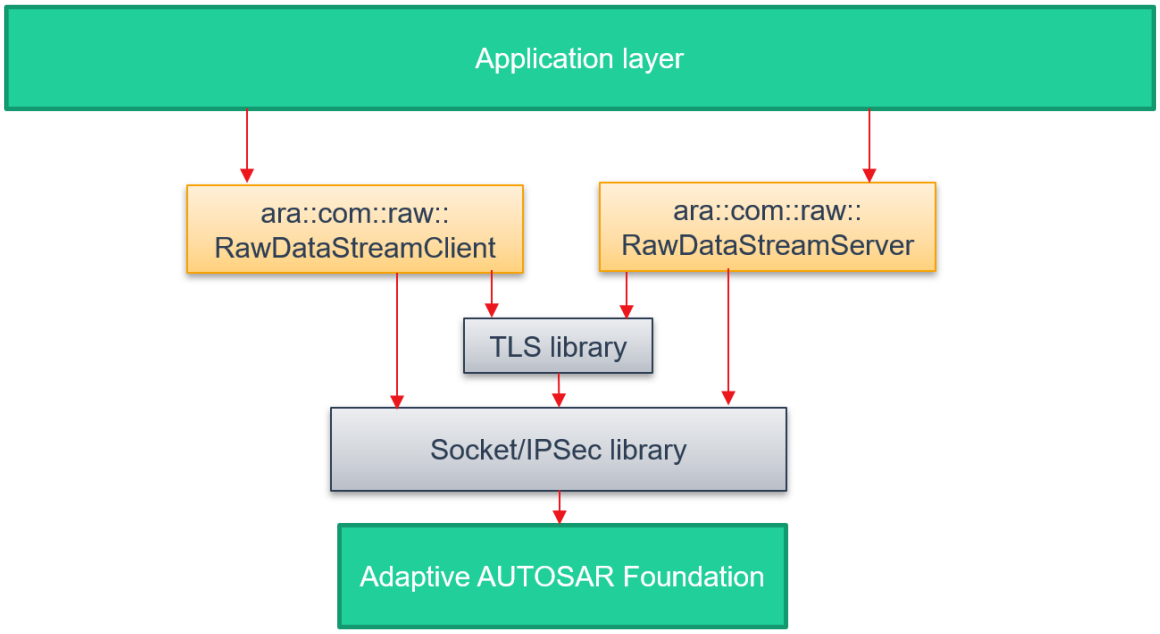


Figure 7.20: Raw Data Stream Logical View.

7.5.1.1 Limitations

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

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

7.5.1.2 Use cases

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

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

RawDataStream socket connections can be setup for UDP or TCP, Unicast or Multicast. Currently the use cases in fig 7.21 are supported.

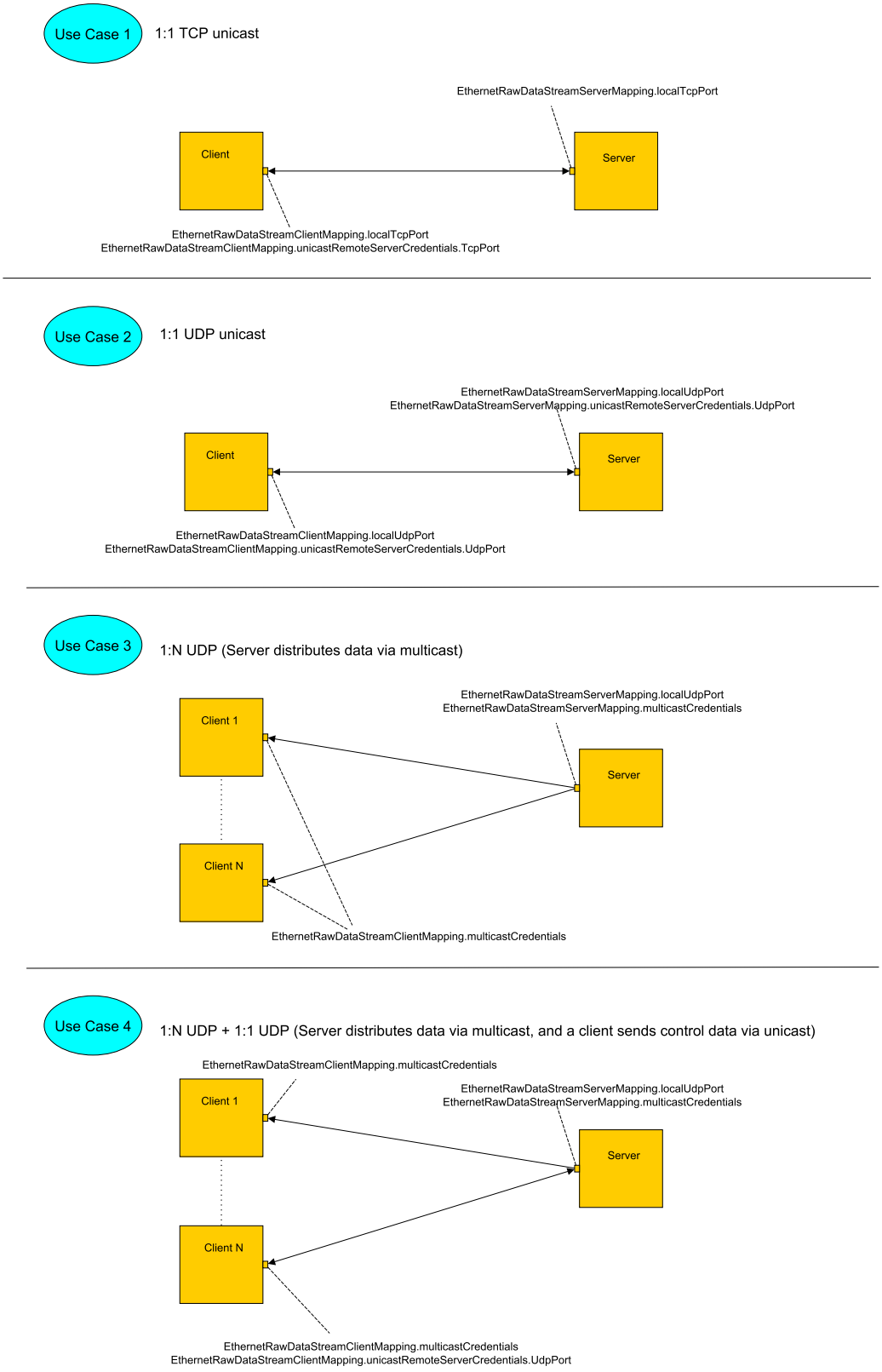


Figure 7.21: The currently supported use cases for Raw Data Streams, and which artifacts in the Deployment model that shall be used to configure the different use cases

7.5.2 Raw Data Streaming

For the Raw Data Stream C++ API reference, see chapter 8.1.3.23.

[SWS_CM_10476] Defining a RawDataStream ⌈To open a `RawDataStream` connection a `RawDataStream` instance is created. The constructor creates the necessary socket data structures for `RawDataStream` Communication, using the artifacts specified in the mapped `EthernetRawDataStreamClientMapping` and `EthernetRawDataStreamServerMapping`.⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_CM_99004]{DRAFT} **Ethernet endpoint configuration** ⌈

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

RawDataStreamClient endpoint and credentials configuration elements:

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

RawDataStreamServer endpoint and credentials configuration elements:

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

- Multicast Credentials: EthernetRawDataStreamServerMapping.multicastCredentials (UDP only)

For the RawDataStreamClients the following shall apply:

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

For the RawDataStreamServers the following shall apply:

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

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

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

[SWS_CM_90216]{DRAFT} **Socket Options configuration** ⌈For both RawDataStreamClients and RawDataStreamServers a list of socket options can be defined in the attribute `socketOption` to be applied to the sockets created for unicast or multicast communication. The options shall be specified as a list of strings. The accepted values are platform specific and shall be documented by the vendor.⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

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

[SWS_CM_90217]{DRAFT} **TLS properties configuration** ⌈For both RawDataStreamClients and RawDataStreamServers (D)TLS properties can be defined in the attributes `tlsSecureComProps` to configure usage of TLS to create secure UDP and TCP channels for the RawDataStreams according to the Transport Layer Security protocol. See [SWS_CM_90211]⌋(*RS_CM_00410*, *RS_CM_00411*)

Note: Usage of (D)TLS is restricted to 1:1 socket connections (use case 1 and 2 of figure Figure 7.21).

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

[SWS_CM_10477] Connect stream link ⌈Each invocation of the Connect operation for a TCP socket connection shall establish a communication link with a remote server that is listening for socket connections, The socket created in the `RawDataStream` instance shall be used for the connection. For UDP socket connections Connect shall do nothing.⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_CM_99005]{DRAFT} **Wait for incoming connections** ⌈Each invocation of the WaitForConnection operation shall wait for and accept incoming requests for establishment of a TCP communication link with a connecting remote client. The socket created and prepared in the RawDataStream instance shall be used for the connection. For UDP socket connections WaitForConnection shall do nothing.⌋(*RS_CM_00411*, *RS_CM_00412*)

[SWS_CM_10478] Shutdown stream link ⌈Each invocation of the Shutdown operation shall destroy the communication link for the stream.⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_CM_10479] Read data from stream ⌈Each invocation of the ReadData operation shall request to read a number of bytes from the stream. The read data shall be moved to a buffer returned as result from the function, together with the actual number of bytes transferred.⌋(*RS_CM_00410*, *RS_CM_00411*, *RS_CM_00412*)

[SWS_CM_10480] Write data to stream ⌈Each invocation of the WriteData operation shall request to write a number of bytes to the stream and send it out on the socket connection. The actual number of bytes transferred shall be returned. It shall be possible to apply a timeout value for the operation. The operation shall write the data to the socket or internal buffer, and then return with the number of bytes written. For efficiency, the Write operation does not wait until data is actually sent on the bus, but the TCP data flow handling shall make sure that data is transmitted and received in the correct order. For UDP connections the order cannot be guaranteed.⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_CM_99006]{DRAFT} **Timeout handling** ⌈For all Connect, WaitForConnection, Read and Write `RawDataStream` operations a timeout value can be specified via a parameter in runtime. If no timeout parameter is given the operation shall block. If a timeout value is specified, and the operation does not finish within the specified time, an error code `RawErrc::kCommunicationTimeout` shall be returned and the technical state of the `RawDataStream` connection shall be restored to the same as before the call was made.⌋*(RS_CM_00410, RS_CM_00411)*

7.6 Communication Group

The Communication Group is a communication concept based on `ara::com` which is designed for Adaptive State Management applications. It can be seen as a composite Service which manages information routing in a defined manner. A Communication Group has one server and multiple clients. The server is able to send broadcast and peer to peer messages to the clients of a Communication Group. The clients can acknowledge these messages. The server of a Communication Group can further verify how many clients are connected to the Communication Group at every time. Applications can connect/disconnect to a Communication Group instance using one of the two Communication Group Service Interfaces, `CommunicationGroupServer` or `CommunicationGroupClient`.

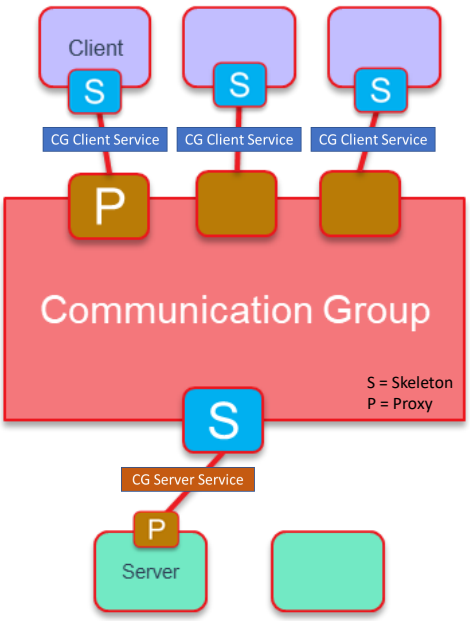


Figure 7.22: Communication Group

7.6.1 Interfaces

The Communication Group uses two Service interfaces, one for a Communication Group Server and one for Communication Group clients.

7.6.1.1 Communication Group Server

[SWS_CM_99000]{DRAFT} **`CommunicationGroupServer` Service** ⌈A Communication Group shall provide a `CommunicationGroupServer` Service to be used by the Server of a Communication Group.⌋*(RS_CM_00600, RS_CM_00601)*

[SWS_CM_99001]{DRAFT} **Broadcast method of `CommunicationGroupServer` Service** ⌈The `CommunicationGroupServer` Service shall provide the method `broadcast` to broadcast messages to the clients of the Communication Group. This method shall take as input parameter the `message` to be broadcasted. In case the `boardcast` method fails the method return shall provide an error code as specified in [SWS_CM_99024].⌋*(RS_CM_00600, RS_CM_00601)*

The C++ signature below presents the resulting `boardcast` method of a generated Service Proxy/Skeleton interface.

```
template <typename T>
ara::core::Future<void> broadcast (const T& msg);
```

[SWS_CM_99002]{DRAFT} **Peer To Peer Message method of `CommunicationGroupServer` Service** ⌈The `CommunicationGroupServer` Service shall provide a method `message` to send a message to a dedicated client of the Communication Group. This method shall take as input parameters the `message` to be sent, and the `clientID` of the client which shall be addressed. In case the `message` method fails the method return shall provide an error code as specified in [SWS_CM_99024].⌋ *(RS_CM_00600, RS_CM_00601)*

The C++ signature below presents the resulting `message` method of a generated Service Proxy/Skeleton interface.

```
template <typename T>
ara::core::Future<void> message (std::uint32_t clientID, const T& msg);
```

[SWS_CM_99014]{DRAFT} **Message Response event of `CommunicationGroupServer` Service** ⌈The `CommunicationGroupServer` Service shall provide an event `response` that contains the respond of a dedicated client to a broadcast or a peer to peer message of the Communication Group. The event shall provide the `response message` and the `clientID` of this response.⌋ *(RS_CM_00600, RS_CM_00601)*

The C++ signature below presents the resulting event `response` message of a generated Service Proxy/Skeleton interface.

```
template <typename R>
struct Response {
 std::uint32_t        clientID;
 const R&         responseMsg
}
```

[SWS_CM_99015]{DRAFT} **List Clients method of `CommunicationGroupServer` Service** ⌈The `CommunicationGroupServer` Service shall provide a method `listClients` to report about the connected clients of the Communication Group. This method shall have no input parameters and shall return the `list of clients`. In case the `listClients` method fails the method return shall provide an error code as specified in [SWS_CM_99024].⌋ *(RS_CM_00600, RS_CM_00601)*

The C++ signature below presents the resulting `listClients` method of a generated Service Proxy/Skeleton interface.

```
ara::core::Future<ara::core::Vector<std::uint32_t>> listClients();
```

7.6.1.2 Communication Group Client

[SWS_CM_99007]{DRAFT} **`CommunicationGroupClient` Service** ⌈The clients of a Communication Group shall provide a `CommunicationGroupClient` Service to be used by a Communication Group.

⌋*(RS_CM_00600, RS_CM_00601)*

[SWS_CM_99008]{DRAFT} **Message method of `CommunicationGroupClient` Service** ⌈The `CommunicationGroupClient` Service shall provide a method `message` for the client to receive a message from the Communication Group. This method shall take as input parameter the `message`. In case the `message` method fails the method return shall provide an error code as specified in [SWS_CM_99024].⌋*(RS_CM_00600, RS_CM_00601)*

The C++ signature below presents the resulting `message` method of a generated Service Proxy/Skeleton interface.

```
template <typename T>
ara::core::Future<void> message (const T& msg);
```

[SWS_CM_99009]{DRAFT} **Message Response event of `CommunicationGroupClient` Service** ⌈The `CommunicationGroupClient` Service shall provide an event `response` for the client to send a response message to the Communication Group. The event shall provide the `response message`.⌋*(RS_CM_00600, RS_CM_00601)*

The C++ signature below presents the resulting event `response` message of a generated Service Proxy/Skeleton interface.

```
template <typename R>
const R& responseMsg;
```

7.6.2 Behavior

The Communication Group performs the following tasks to enable a Communication Group.

[SWS_CM_99010]{DRAFT} **Broadcast task** ⌈A Broadcast task shall be triggered by the `broadcast` method of the `CommunicationGroupServer` Service. The CommunicationGroup shall forward this broadcast message to all connected clients by calling the `message` method of the `CommunicationGroupClient` Service from each connected client.⌋*(RS_CM_00600, RS_CM_00601)*

[SWS_CM_99011]{DRAFT} **Peer To Peer message task** ⌈A Peer to Peer message task shall be triggered by the `message` method (which includes the client address) of the `CommunicationGroupServer` Service. The CommunicationGroup shall forward

this message to the addressed client by calling the `message` method of the `CommunicationGroupClient` Service of this client.⌋*(RS_CM_00600, RS_CM_00601)*

[SWS_CM_99012]{DRAFT} **Message Response task** ⌈The Message Response task shall be triggered by the `message response` event of the `CommunicationGroupClient` Service from a client . The CommunicationGroup shall forward this response message with the client source address to the `message response` event of the `CommunicationGroupServer` Service.⌋*(RS_CM_00600, RS_CM_00601)*

[SWS_CM_99013]{DRAFT} **List Clients task** ⌈The List Clients task shall be triggered by the `list clients` method of the `CommunicationGroupServer` Service. The CommunicationGroup shall provide the list of all connected client addresses with the return of the `list clients` method of the `CommunicationGroupServer` Service.⌋ *(RS_CM_00600, RS_CM_00601)*

7.6.3 Connection

The connection and disconnection to Communication Group is performed by standard `ara::com` functions.

7.6.3.1 Communication Group Server

The Server of a Communication Group connects to a Communication Group by connecting to the `CommunicationGroupServer` Service of this Communication group (using `FindService` or `StartFindService`).

[SWS_CM_99016]{DRAFT} **Connection Status of a Communication Group Server** ⌈The Server of the Communication Group shall be considered to be connected if the server has successfully subscribed to the response `message response` event of the `CommunicationGroupServer` Service, else the Server shall be considered not connected.⌋*(RS_CM_00600, RS_CM_00601)*

7.6.3.2 Communication Group Client

A Communication Group client connects to a Communication Group by offering the `CommunicationGroupClient` Service. A Communication Group client disconnects to a Communication Group by stop offering the `CommunicationGroupClient` Service.

7.6.4 Limitations

The Communication Group concept has the following limitations:

- There is only one Server for an instance of a Communication Group at a given time.
- A Client provides the `CommunicationGroupClient` Service to only one instance of a Communication Group at a given time.

The figure below outlines a connection example for a Communication Group.

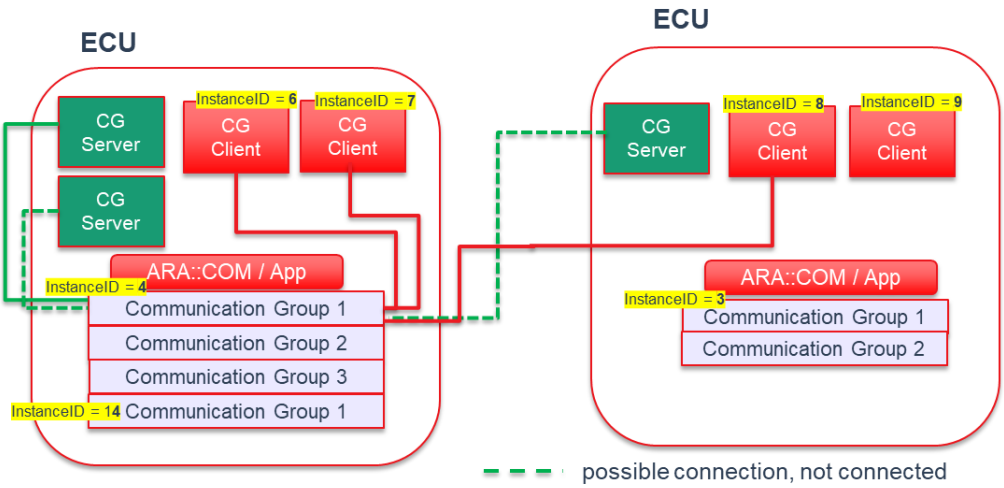


Figure 7.23: Communication Group connection example

7.6.5 Communication Group Model

The model of Communication Group is labeled by one of three standard `ServiceInterface.category` values. See also the *TPS_ManifestSpecification* [6].

[SWS_CM_99017]{DRAFT} **`category` value `COMMUNICATION_GROUP`** ⌈The `ServiceInterface.category` value `COMMUNICATION_GROUP` shall be used to define a Communication Group `template`.⌋*(RS_CM_00600, RS_CM_00601)*

[SWS_CM_99018]{DRAFT} **`category` value `COMMUNICATION_GROUP_SERVER`** ⌈The `ServiceInterface.category` value `COMMUNICATION_GROUP_SERVER` shall be used to define `CommunicationGroupServer` service.⌋*(RS_CM_00600, RS_CM_00601)*

[SWS_CM_99019]{DRAFT} **`category` value `COMMUNICATION_GROUP_CLIENT`** ⌈The `ServiceInterface.category` value `COMMUNICATION_GROUP_CLIENT` shall be used to define `CommunicationGroupClient` service.⌋*(RS_CM_00600, RS_CM_00601)*

The figure below presents the relations between these `category` values.

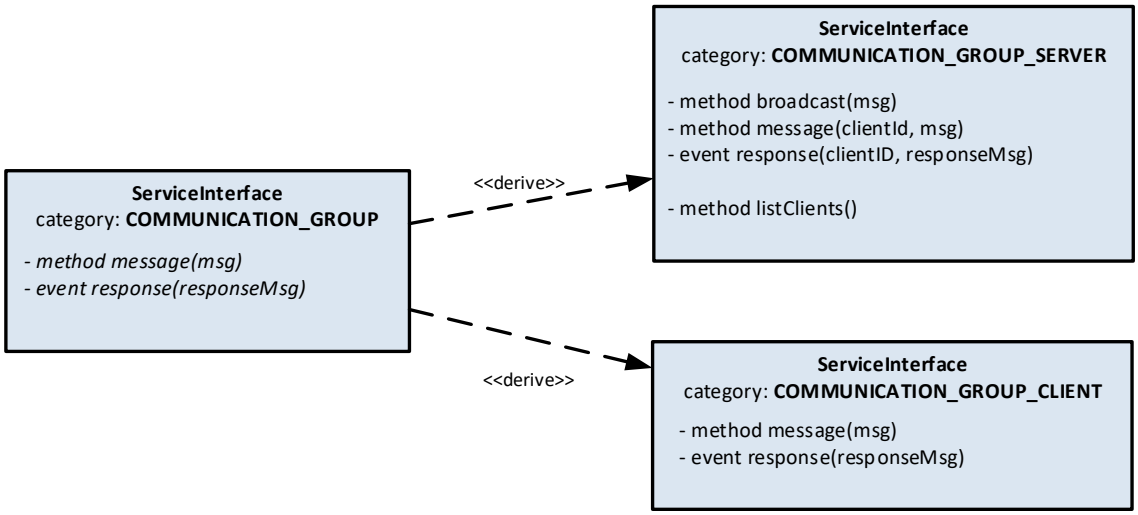


Figure 7.24: Communication Group service interface categories

7.6.6 Communication Group Creation

A Communication Group is created by defining a Communication Group `template` only.

[SWS_CM_99020]{DRAFT} **Communcation Group `template`** ⌈The Communication Group `template` is a `ServiceInterface` of type `CommunicationGroupClient` with the `category` value `COMMUNICATION_GROUP`. It shall be used to define a Communication Group, where:

- The `SHORT-NAME` of this `template` shall define of the name of the Communication Group.
- The event definition according to [SWS_CM_99009] shall define the data type of the `message responses` of the Communication Group.
- The method definition according to [SWS_CM_99008] shall define the data type of the `messages` of the Communication Group.

⌋*(RS_CM_00600, RS_CM_00601)*

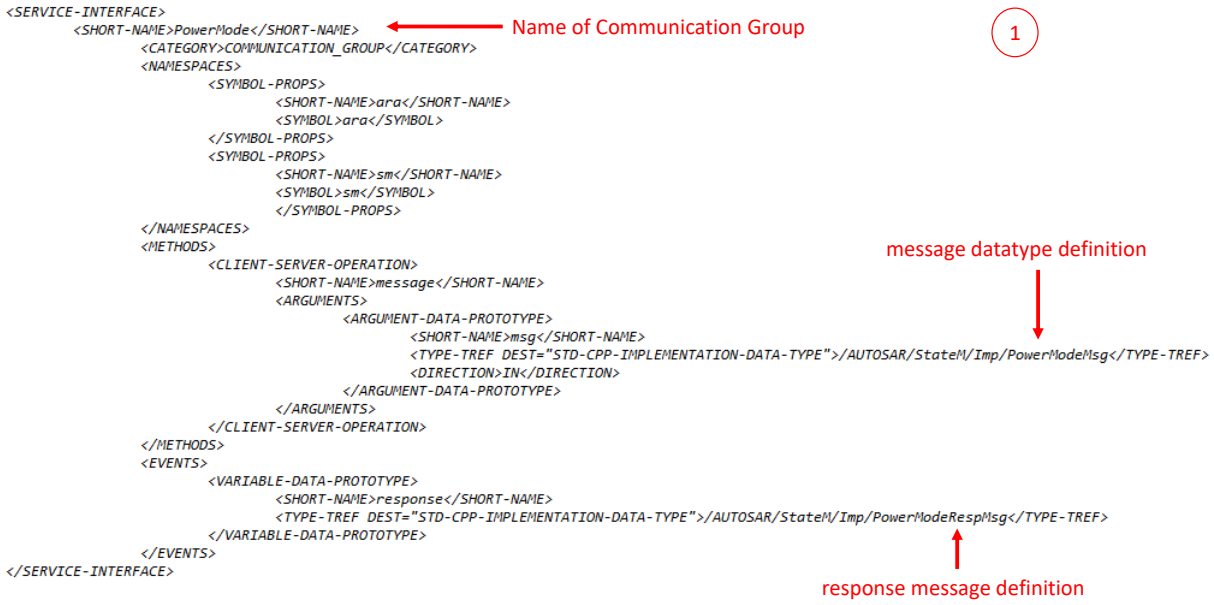
Based on the Communication Group `template` [SWS_CM_99020] the `ServiceInterfaces` for the `CommunicationGroupServer` [SWS_CM_99000] and the `CommunicationGroupClient` [SWS_CM_99007] can be generated/derived.

[SWS_CM_99021]{DRAFT} **SHORT-NAME value of generated Communication-GroupServer service** ⌈The SHORT-NAME value of generated Communication-GroupServer service shall be the SHORT-NAME of the according Communication Group template concatenated by the name Server.⌋*(RS_CM_00600, RS_CM_-00601)*

[SWS_CM_99022]{DRAFT} **SHORT-NAME value of generated Communication-GroupClient service** ⌈The SHORT-NAME value of generated Communication-GroupClient service shall be the SHORT-NAME of the according Communication Group template concatenated by the name Client.⌋*(RS_CM_00600, RS_CM_-00601)*

The figures below outline the Communication Group creation flow.

The Communication Group Template defines the name of the Communication Group and defines the message and message response datatypes.




```
<SERVICE-INTERFACE>
        <SHORT-NAME>PowerMode</SHORT-NAME>
                <CATEGORY>COMMUNICATION_GROUP</CATEGORY>
                <NAMESPACES>
                        <SYMBOL-PROPS>
                                <SHORT-NAME>ara</SHORT-NAME>
                                <SYMBOL>ara</SYMBOL>
                        </SYMBOL-PROPS>
                        <SYMBOL-PROPS>
                                <SHORT-NAME>sm</SHORT-NAME>
                                <SYMBOL>sm</SYMBOL>
                                </SYMBOL-PROPS>
                </NAMESPACES>
                <METHODS>
                        <CLIENT-SERVER-OPERATION>
                                <SHORT-NAME>message</SHORT-NAME>
                                <ARGUMENTS>
                                        <ARGUMENT-DATA-PROTOTYPE>
                                                <SHORT-NAME>msg</SHORT-NAME>
                                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StateM/Imp/PowerModeMsg</TYPE-TREF>
                                                <DIRECTION>IN</DIRECTION>
                                        </ARGUMENT-DATA-PROTOTYPE>
                                </ARGUMENTS>
                        </CLIENT-SERVER-OPERATION>
                </METHODS>
                <EVENTS>
                        <VARIABLE-DATA-PROTOTYPE>
                                <SHORT-NAME>response</SHORT-NAME>
                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StateM/Imp/PowerModeRespMsg</TYPE-TREF>
                        </VARIABLE-DATA-PROTOTYPE>
                </EVENTS>
</SERVICE-INTERFACE>
```

Figure 7.25: Communication Group Template

The CommunicationGroupServer and the CommunicationGroupClient Service descriptions are derived from the Communication Group Template.

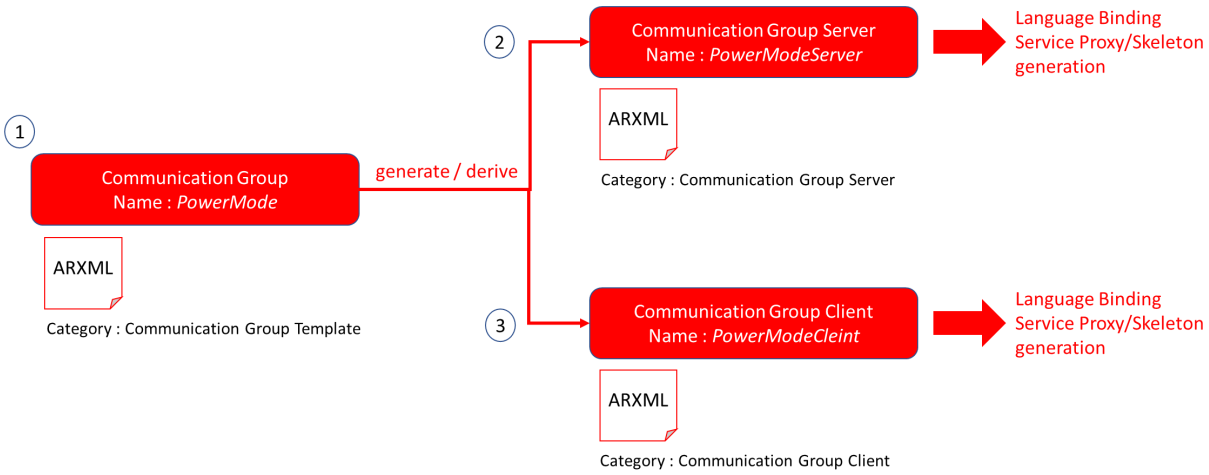


Figure 7.26: Communication Group Flow

```
<SERVICE-INTERFACE>
        <SHORT-NAME>PowerModeServer</SHORT-NAME>
                <CATEGORY>COMMUNICATION_GROUP_SERVER</CATEGORY>
                <NAMESPACES>
                        <SYMBOL-PROPS>
                                <SHORT-NAME>ara</SHORT-NAME>
                                <SYMBOL>ara</SYMBOL>
                        </SYMBOL-PROPS>
                        <SYMBOL-PROPS>
                                <SHORT-NAME>sm</SHORT-NAME>
                                <SYMBOL>sm</SYMBOL>
                                </SYMBOL-PROPS>
                </NAMESPACES>
                <METHODS>
                        <CLIENT-SERVER-OPERATION>
                                <SHORT-NAME>broadcast</SHORT-NAME>
                                <ARGUMENTS>
                                        <ARGUMENT-DATA-PROTOTYPE>
                                                <SHORT-NAME>msg</SHORT-NAME>
                                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StateM/Imp/PowerModeMsg</TYPE-TREF>
                                                <DIRECTION>IN</DIRECTION>
                                        </ARGUMENT-DATA-PROTOTYPE>
                                </ARGUMENTS>
                        </CLIENT-SERVER-OPERATION>
                        <CLIENT-SERVER-OPERATION>
                                <SHORT-NAME>message</SHORT-NAME>
                                <ARGUMENTS>
                                        <ARGUMENT-DATA-PROTOTYPE>
                                                <SHORT-NAME>clientID</SHORT-NAME>
                                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StdTypes/uint32_t</TYPE-TREF>
                                                <DIRECTION>IN</DIRECTION>
                                        </ARGUMENT-DATA-PROTOTYPE>
                                </ARGUMENTS>
                                <ARGUMENTS>
                                        <ARGUMENT-DATA-PROTOTYPE>
                                                <SHORT-NAME>msg</SHORT-NAME>
                                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StateM/Imp/PowerModeMsg</TYPE-TREF>
                                                <DIRECTION>IN</DIRECTION>
                                        </ARGUMENT-DATA-PROTOTYPE>
                                </ARGUMENTS>
                        </CLIENT-SERVER-OPERATION>
                        <CLIENT-SERVER-OPERATION>
                                <SHORT-NAME>listClients</SHORT-NAME>
                                <ARGUMENTS>
                                        <ARGUMENT-DATA-PROTOTYPE>
                                                <SHORT-NAME>clients</SHORT-NAME>
                                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StateM/Imp/Clients</TYPE-TREF>
                                                <DIRECTION>IN</DIRECTION>
                                        </ARGUMENT-DATA-PROTOTYPE>
                                </ARGUMENTS>
                        </CLIENT-SERVER-OPERATION>
                </METHODS>
                <EVENTS>
                        <VARIABLE-DATA-PROTOTYPE>
                                <SHORT-NAME>response</SHORT-NAME>
                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StateM/Imp/PowerModeResponse</TYPE-TREF>
                        </VARIABLE-DATA-PROTOTYPE>
                </EVENTS>
</SERVICE-INTERFACE>
```

2

Figure 7.27: Communication Group Server Service Description

Note:

- The `PowerModeResponse` datatype is a structure of `clientID` and `PowerModeRespMsg` datatype.
- The `Clients` datatype is a vector of `uint32` datatype.

```
<SERVICE-INTERFACE>
        <SHORT-NAME>PowerModeClient</SHORT-NAME>
                <CATEGORY>COMMUNICATION_GROUP_CLIENT</CATEGORY>
                <NAMESPACES>
                        <SYMBOL-PROPS>
                                <SHORT-NAME>ara</SHORT-NAME>
                                <SYMBOL>ara</SYMBOL>
                        </SYMBOL-PROPS>
                        <SYMBOL-PROPS>
                                <SHORT-NAME>sm</SHORT-NAME>
                                <SYMBOL>sm</SYMBOL>
                                </SYMBOL-PROPS>
3               </NAMESPACES>
                <METHODS>
                        <CLIENT-SERVER-OPERATION>
                                <SHORT-NAME>message</SHORT-NAME>
                                <ARGUMENTS>
                                        <ARGUMENT-DATA-PROTOTYPE>
                                                <SHORT-NAME>msg</SHORT-NAME>
                                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StateM/Imp/PowerModeMsg</TYPE-TREF>
                                                <DIRECTION>IN</DIRECTION>
                                        </ARGUMENT-DATA-PROTOTYPE>
                                </ARGUMENTS>
                        </CLIENT-SERVER-OPERATION>
                </METHODS>
                <EVENTS>
                        <VARIABLE-DATA-PROTOTYPE>
                                <SHORT-NAME>response</SHORT-NAME>
                                <TYPE-TREF DEST="STD-CPP-IMPLEMENTATION-DATA-TYPE">/AUTOSAR/StateM/Imp/PowerModeRespMsg</TYPE-TREF>
                        </VARIABLE-DATA-PROTOTYPE>
                </EVENTS>
</SERVICE-INTERFACE>
```

Figure 7.28: Communication Group Client Service Description

7.7 Optional Execution Context

Some `ara::com` API's with an asynchronous callback allow the use of an optional execution context parameter (see [SWS_CM_11352], [SWS_CM_11352], [SWS_CM_11354], [SWS_CM_11356], [SWS_CM_11358], [SWS_CM_11360], [SWS_CM_11362]). The execution context parameter gives the user more control over the execution environment of a method call.

[SWS_CM_11364]{DRAFT} **Minimal behaviour of provided Execution Context** ⌈An optionally provided execution context executor shall:

- execute every function that it was passed to.
- execute each function it was passed to only once.

⌋*(RS_CM_00204)*

7.8 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]{DRAFT} **Change of Service Interface Deployment** ⌈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`.

⌋*(RS_CM_00315)*
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_10385]{DRAFT} **Change of Service Instance Deployment** ⌈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`.

⌋*(RS_CM_00315)* 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]{DRAFT} **Change of Network Configuration** ⌈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`.

⌋*(RS_CM_00315)*

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]{DRAFT} **Abstract Network Protocol Binding** ⌈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 `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).⌋ *(RS_CM_00200, RS_AP_00137)*

[SWS_CM_10416]{DRAFT} **Reception of a malformed message** ⌈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.

7.8.1 SOME/IP Network binding

SOME/IP supports different kind of bindings:

SOME/IP Events:

- uni-cast 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 uni-cast to multi-cast to gain a better network utilization. Below the multi-cast-threshold SOME/IP is maintaining for a subscription a single uni-cast communication.

SOME/IP Events:

- many-to-one communication using multiple uni-cast communications

[SWS_CM_10000] SOME/IP Compliance ⌈The SOME/IP network binding shall implement the SOME/IP Protocol and the SOME/IP Service Discovery Protocol defined in [5] and [12].⌋*(RS_CM_00204, RS_CM_00205)*

[SWS_CM_10013] Header Byte order ⌈All headers shall be encoded in network byte order Big Endian (MostSignificantByteFirst) [RFC 791].⌋*(RS_CM_00204, RS_SOMEIP_00026)*

[SWS_CM_10172] Payload Byte order definition ⌈The byte order of the parameters inside the payload shall be defined by byteOrder of ApSomeipTransformationProps.⌋*(RS_CM_00204, RS_SOMEIP_00026)*

[SWS_CM_10240]{DRAFT} **Session handling state** ⌈If ApSomeipTransformationProps.sessionHandling is present and set to value SOMEIPTransformerSessionHandlingEnum.sessionHandlingActive, the Session handling shall be Active. If ApSomeipTransformationProps.sessionHandling is present and set to value SOMEIPTransformerSessionHandlingEnum.sessionHandlingInactive, the Session handling shall be Inactive.⌋*(RS_CM_00204, RS_SOMEIP_00012)*

7.8.1.1 Service Discovery

[SWS_CM_00201] Start of service discovery protocol on Server side ⌈The registration of a new offered service which is bound to SOME/IP shall trigger the start of the initial wait phase of the SOME/IP service discovery protocol.⌋*(RS_CM_00204, RS_CM_00101, RS_SOMEIPSD_00024, RS_SOMEIPSD_00013)*

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

- [TPS_MANI_03012] (Initial Wait Phase),
- [TPS_MANI_03013] (Repetition Wait Phase),
- [TPS_MANI_03014] (Main Phase).

The corresponding timing parameters for these phases are configured via InitialSdDelayConfig and RequestResponseDelay. The sharing of timers is described in [TPS_MANI_03230].

[SWS_CM_00209] Start of service discovery protocol on Client side ⌈The search for a new service which is bound to SOME/IP shall trigger the start of the initial wait phase.⌋ *(RS_CM_00204, RS_CM_00102, RS_SOMEIPSD_00024, RS_SOMEIPSD_00008)*

(See also [PRS_SOMEIPSD_00395], [PRS_SOMEIPSD_00397], [PRS_SOMEIPSD_00399], [PRS_SOMEIPSD_00416], [PRS_SOMEIPSD_00435], [PRS_SOMEIPSD_00752] and [PRS_SOMEIPSD_00133])

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

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

The corresponding timing parameters for these phases are configured via `InitialSdDelayConfig` and `RequestResponseDelay`. The sharing of timers is described in [TPS_MANI_03231]. TTL for Find Service Entries is described in [TPS_MANI_03028].

[SWS_CM_00202] SOME/IP FindService message ⌈The entries in the SOME/IP FindService message shall be as follows:

- The entry type shall be set to FindService (see [PRS_SOMEIPSD_00268] for numerical value).
- The Service ID shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Instance ID shall be 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.
- Major Version of the `RequiredSomeipServiceInstance` that is searched shall be derived from the Manifest where the `SomeipServiceVersion` element that is aggregated by the `SomeipServiceInterfaceDeployment` in the role `serviceInterfaceVersion` defines the `majorVersion`.
- Minor Version of the `RequiredSomeipServiceInstance` that is searched shall be derived from the Manifest from the `requiredMinorVersion` attribute in the `RequiredSomeipServiceInstance`.

  if `versionDrivenFindBehavior` is set to `minimumMinorVersion` then the `minorVersion` 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 minorVersion shall be set with the requiredMinorVersion. If the minorVersion is set to "ALL", then 0xFFFF FFFF shall be used.

- TTL shall be derived from the Manifest where the SomeipSdClientServiceInstanceConfig element that is referenced by the RequiredSomeipServiceInstance in the role sdClientConfig defines the serviceFindTimeToLive.
- 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.

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00102*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00005*, *RS_SOMEIPSD_00008*, *RS_SOMEIPSD_00010*)

[SWS_CM_10202] Version blacklist ⌈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.blacklistedVersion.⌋ (*RS_CM_00701*)

[SWS_CM_00203] SOME/IP OfferService message ⌈The entries in the SOME/IP OfferService message shall be as follows:

- The entry type shall be set to OfferService (see [PRS_SOMEIPSD_00268] for numerical value).
- The Service ID shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Instance ID shall be 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 of the SomeipServiceInterfaceDeployment that is offered shall be derived from the Manifest where the SomeipServiceVersion element that is aggregated by the SomeipServiceInterfaceDeployment in the role serviceInterfaceVersion defines the majorVersion.
- Minor Version of the SomeipServiceInterfaceDeployment that is offered shall be derived from the Manifest where the SomeipServiceVersion element that is aggregated by the SomeipServiceInterfaceDeployment in the role serviceInterfaceVersion defines the minorVersion.
- TTL shall be derived from the Manifest where the SomeipSdServerServiceInstanceConfig element that is referenced by the ProvidedSomeipServiceInstance in the role sdServerConfig defines the serviceOfferTimeToLive.

- 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.
- 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 if SomeipServiceInstanceToMachineMapping.tcpPort is configured.

  In case the port number (SomeipServiceInstanceToMachineMapping.udpPort or SomeipServiceInstanceToMachineMapping.tcpPort) is cofigured 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.
- Configuration Option shall be used in the offer message if at least one capabilityRecord is defined for the ProvidedSomeipServiceInstance. The content of the Configuration Option shall be derived from the key/value pairs defined in each capabilityRecord.

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00101*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00005*, *RS_SOMEIPSD_00010*, *RS_SOMEIPSD_00013*, *RS_SOMEIPSD_00025*)

[SWS_CM_00204] SOME/IP StopOffer message ⌈The entries in the SOME/IP StopOffer message shall be as follows:

- The entry type shall be set to StopOfferService (see [PRS_SOMEIPSD_00268] for numerical value).
- ServiceId shall be set to the same value as in the OfferService message.
- InstanceId shall be set to the same value as in the OfferService message.
- Major Version shall be set to the same value as in the OfferService message.
- Minor Version shall be set to the same value as in the OfferService message.

- TTL shall be set to 0x000000 value.
- 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.

⌋(*RS_CM_00204*, *RS_CM_00105*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00005*, *RS_SOMEIPSD_00010*, *RS_SOMEIPSD_00014*)

[SWS_CM_10377] Sending SOME/IP SubscribeEventgroup messages - initial ⌈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).⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00103*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00015*)

[SWS_CM_10381] Sending SOME/IP SubscribeEventgroup messages - renewal ⌈If the TTL of an active subscription for a particular `Eventgroup` is about to expire 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`.⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00103*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00015*)

[SWS_CM_00205] Content of SOME/IP SubscribeEventgroup message ⌈The entries in the SOME/IP SubscribeEventgroup message shall be as follows:

- The entry type shall be set to SubscribeEventgroup (see [PRS_SOMEIPSD_00270] for numerical value).
- 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.

- Eventgroup ID 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.
- TTL shall be derived from Manifest where the RequiredSomeipServiceInstance element aggregates the SomeipRequiredEventGroup in the role requiredEventGroup. The SomeipRequiredEventGroup aggregates the sdClientEventGroupTimingConfig where the timeToLive 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 theIpv4Configuration 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 theIpv6Configuration 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.
  - UDP shall be used if SomeipServiceInstanceToMachineMapping.udpPort is configured and the SomeipEventGroup contains SomeipEventDeployments where the transportProtocol is set to udp. The UDP port shall be derived from SomeipServiceInstanceToMachineMapping.udpPort. In case the port number (SomeipServiceInstanceToMachineMapping.udpPort) is cofigured 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.
  - TCP shall be used if SomeipServiceInstanceToMachineMapping.tcpPort is configured and the SomeipEventGroup contains SomeipEventDeployments where the transportProtocol is set

to tcp. The TCP port shall be derived from SomeipServiceInstanceToMachineMapping.tcpPort. In case the port number (SomeipServiceInstanceToMachineMapping.tcpPort) is cofigured 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.

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00103*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00005*, *RS_SOMEIPSD_00015*)

[SWS_CM_00206] SOME/IP SubscribeEventgroupAck message ⌈The entries in the SOME/IP SubscribeEventgroupAck message shall be as follows:

- The entry type shall be set to SubscribeEventgroupAck (see [PRS_SOMEIPSD_00270] for numerical value).
- ServiceId shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- InstanceId shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- Major Version shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- Eventgroup ID shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- TTL shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupAck message.
- 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.
- 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.

⌋(*RS_CM_00204*, *RS_SOMEIPSD_00015*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00002*, *RS_SOMEIPSD_00003*, *RS_SOMEIPSD_00005*)

[SWS_CM_00208] SOME/IP SubscribeEventgroupNack message ⌈The entries in the SOME/IP SubscribeEventgroupNack message shall be as follows:

- The entry type shall be set to SubscribeEventgroupNack (see [PRS_SOMEIPSD_00270] for numerical value).
- ServiceId shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- InstanceId shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- Major Version shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- Eventgroup ID shall be set to the same value as in the SubscribeEventgroup message that is answered by this SubscribeEventgroupNack message.
- TTL shall be set to the 0x000000 value.

⌋(*RS_CM_00204*, *RS_SOMEIPSD_00016*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00005*)

[SWS_CM_10378] Sending SOME/IP StopSubscribeEventgroup messages ⌈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.⌋(*RS_CM_00204*, *RS_CM_00104*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00005*, *RS_SOMEIPSD_00017*)

[SWS_CM_00207] Content of SOME/IP StopSubscribeEventgroup message ⌈The entries in the SOME/IP StopSubscribeEventgroup message shall be as follows:

- The entry type shall be set to StopSubscribeEventgroup (see [PRS_SOMEIPSD_00270] for numerical value).
- ServiceId shall be set to the same value as in the SubscribeEventgroup message.
- InstanceId shall be set to the same value as in the SubscribeEventgroup message.
- Major Version shall be set to the same value as in the SubscribeEventgroup message.
- Eventgroup ID shall be set to the same value as in the SubscribeEventgroup message.
- TTL shall be set to the 0x000000 value.

- 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.

⌋(*RS_CM_00204*, *RS_CM_00104*, *RS_SOMEIPSD_00006*, *RS_SOMEIPSD_00005*, *RS_SOMEIPSD_00017*)

7.8.1.2 Accumulation of SOME/IP messages

[SWS_CM_10387] Data accumulation for UDP data transmission ⌈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.⌋(*RS_CM_00204*)

[SWS_CM_10388] Enabling of data accumulation for UDP data transmission ⌈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.⌋(*RS_CM_00204*)

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

⌋(*RS_CM_00204*)

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

⌋*(RS_CM_00204)*

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].

7.8.1.3 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 (see [SWS_CM_00701]) 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 seperate 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 (see [SWS_CM_00701]) method is not allowed, since according to [SWS_CM_00181] the EventReceiveHandlershall use the GetNewSamples method to access the retrieved event data.

[SWS_CM_11270]{DRAFT} **Selecting elements of the ServiceInterface for SecOC transmission** ⌈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 ServiceInterfaceElementSecureComConfigthat 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.

⌋(*RS_SEC_04001, RS_SEC_04003*)

7.8.1.4 Handling Events

[SWS_CM_10287] Conditions for sending of a SOME/IP event message ⌈The sending of a SOME/IP event message shall be requested by invoking the Send method of the respective Event class (see [SWS_CM_00162] and [SWS_CM_90437]) 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). An active subscriber is an adaptive application that has invoked the Subscribe method of the respective Event class (see [SWS_CM_00141]) and has not canceled the subscription by invoking the Unsubscribe method of the respective Event class (see [SWS_CM_00151]) and where the subscription has not yet expired since the TTL contained in the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has been exceeded.⌋(*RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017*)

[SWS_CM_10288] Transport protocol for sending of a SOME/IP event message ⌈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.eventDeployment.transportProtocol in the Manifest if this threshold has not been reached (see [PRS_SOMEIPSD_00802]).⌋(*RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00010*)

[SWS_CM_10289] Source of a SOME/IP event message ⌈The SOME/IP event message shall use the unicast IP address and port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00304]) of the SOME/IP OfferService message ([SWS_CM_00203]) as source address and source port for the transmission.⌋(*RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042*)

[SWS_CM_10290] Destination of a SOME/IP event message ⌈The SOME/IP event message shall use the multicast IP address and the port taken from the IPv4/v6 Multicast Option (see [PRS_SOMEIPSD_00322]) 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 SOME/IP event message shall use the unicast IP address and the port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00304]) 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.⌋(*RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042*)

[SWS_CM_10291] Content of the SOME/IP event message ⌈The entries in the SOME/IP event message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines 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 information whether the Session Handling is activated or deactivated for an event can be derived from the `sessionHandling` attribute contained in the `SomeipTransformationProps` that is referenced by the `TransformationPropsToServiceInterfaceElementMapping` that in turn points to the event.

- 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.

⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004)* The serialization rules are explained in section 7.8.1.8.

[SWS_CM_10292] Checks for a received SOME/IP event message ⌈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_00040]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element in the Manifest to determine the right `ServiceInterface`.
- Verify that the Event ID (see [PRS_SOMEIP_00040]) matches 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).⌋*(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)*

[SWS_CM_10293] Identifying the right event ⌈Using the Service ID (see [PRS_SOMEIP_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element as well as the Event ID (see [PRS_SOMEIP_00040]) and the eventId attribute of the SomeipEventDeployments of the SomeipServiceInterfaceDeployment, the right event shall be identified.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00022)*

[SWS_CM_10379] Silently discarding SOME/IP event messages for unsubscribed events ⌈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, the received SOME/IP event message shall be silently discarded (i.e., [SWS_CM_10294], [SWS_CM_10295], and [SWS_CM_10296] shall *not* be performed).⌋*(RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004)*

[SWS_CM_10296] Invoke receive handler ⌈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.⌋*(RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004)*

[SWS_CM_10294] Deserializing the payload ⌈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 deserialized according to the SOME/IP serialization rules.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00028)* The serialization rules are explained in section 7.8.1.8.

[SWS_CM_10295] Providing the received event data ⌈The deserialized payload containing the event data shall be provided via the GetNewSamples (see [SWS_CM_00701]) method of the respective Event class for the event determined

according to [SWS_CM_10293].⌋*(RS_CM_00204, RS_CM_00202, RS_SOMEIP_00004)*

[SWS_CM_10360]{DRAFT} **Failures in sending a SOME/IP event message** ⌈If the sending of the SOME/IP event message 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 Send() method of the respective Event class (see [SWS_CM_00162] and [SWS_CM_90437]).⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_CM_00004)*

7.8.1.5 Handling Triggers

[SWS_CM_10511]{DRAFT} **Conditions for sending of a SOME/IP trigger** ⌈The sending of a SOME/IP 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). An active subscriber is an adaptive application that has invoked the Subscribe method of the respective Trigger class (see [SWS_CM_00723]) and has not canceled the subscription by invoking the Unsubscribe method of the respective Trigger class (see [SWS_CM_00810]) and where the subscription has not yet expired since the TTL contained in the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has been exceeded.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017)*

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 7.8.1.4 are also valid for Triggers since a trigger defines a special kind of an event.

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

[SWS_CM_10512]{DRAFT} **Content of the SOME/IP trigger** ⌈The entries in the SOME/IP trigger shall be as follows:

- The Service ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.

- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines 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 information whether the Session Handling is activated or deactivated for a trigger can be derived from the sessionHandling attribute contained in the ApSomeipTransformationProps that is referenced by the TransformationPropsToServiceInterfaceElementMapping that in turn points to the trigger.
- 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 triggers and thus (according to [PRS_SOMEIP_00925]) shall be set to E_OK (0x00).

⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004)*

[SWS_CM_10513]{DRAFT} **Checks for a received SOME/IP trigger** ⌈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_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.

- Verify that the Event ID (see [PRS_SOMEIP_00040]) matches 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).⌋*(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)*

[SWS_CM_10514]{DRAFT} **Identifying the right trigger** ⌈Using the Service ID (see [PRS_SOMEIP_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element as well as the Event ID (see [PRS_SOMEIP_00040]) and the eventId attribute of the SomeipEventDeployments of the SomeipServiceInterfaceDeployment, the right trigger shall be identified.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00022)*

[SWS_CM_10515]{DRAFT} **Silently discarding SOME/IP triggers for unsubscribed triggers** ⌈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 SubscribeTriggergroup message (see [SWS_CM_00205]) has expired, the received SOME/IP trigger shall be silently discarded (i.e., [SWS_CM_00226], and [SWS_CM_00249] shall *not* be performed).⌋*(RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004)*

[SWS_CM_10516]{DRAFT} **Invoke receive handler** ⌈In case a receive handler was registered using the SetReceiveHandler method (see [SWS_CM_00249]) of the respective Trigger class for the trigger determined according to [SWS_CM_10514] this registered receive handler shall be invoked.⌋*(RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004)*

[SWS_CM_10517]{DRAFT} **Failures in sending a SOME/IP trigger** ⌈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 an error indicating "network binding failure" in the Result of the Send() method of the respective Trigger class (see [SWS_CM_00721]).⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_CM_00004)*

7.8.1.6 Handling Method Calls

[SWS_CM_10297] Conditions for sending of a SOME/IP request message ⌈The sending of a SOME/IP request message shall be requested by invoking the function call operator (`operator()`) of the respective `Method` class (see [SWS_CM_00196]) 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).⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007)*

[SWS_CM_10441] Failures in sending of a SOME/IP request message ⌈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 `Future` returned by the function call operator (`operator()`) of the respective `Method` class (see [SWS_CM_00196]) ready according to [SWS_CM_10440].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007)*

[SWS_CM_10298] Transport protocol for sending of a SOME/IP request message ⌈The SOME/IP request message shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.methodDeployment.transportProtocol` in the Manifest.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00010)*

[SWS_CM_10299] Source of a SOME/IP request message ⌈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.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00010)*

[SWS_CM_10300] Destination of a SOME/IP request message ⌈The SOME/IP request message shall use the unicast IP address and port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00304]) of the SOME/IP OfferService message ([SWS_CM_00203]) 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.⌋(*RS_CM_00204*, *RS_CM_00212*, *RS_CM_00213*, *RS_SOMEIP_00006*, *RS_SOMEIP_00007*)

[SWS_CM_10301] Content of the SOME/IP request message ⌈The entries in the SOME/IP request message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00038]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00038]) 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 achived 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 `ArgumentDataPrototypes` of the `ClientServerOperation` with `direction` set to `in` and `inout` serialized according to their order) according to the SOME/IP serialization rules.

⌋(*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 SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10302] Checks for a received SOME/IP request message ⌈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_00040]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element in the Manifest to determine the right `ServiceInterface`.
- Verify that the Method ID (see [PRS_SOMEIP_00038]) matches the `methodId` attribute of one of the `SomeipMethodDeployment`s 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).⌋ *(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)*

[SWS_CM_10303] Identifying the right method ⌈Using the Service ID (see [PRS_SOMEIP_00040]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element as well as the Method ID (see [PRS_SOMEIP_00038]) and the `methodId` attribute of the `SomeipMethodDeployment`s of the `SomeipServiceInterfaceDeployment`, the right method shall be identified.⌋ *(RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00021)*

[SWS_CM_10304] Deserializing the payload ⌈Based on the method determined according to [SWS_CM_10303] the Payload of the SOME/IP request message shall be deserialized according to the SOME/IP serialization rules.⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00028)* The SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10306] Invoke the method - event driven ⌈In case a `MethodCallProcessingMode` of either `kEvent` or `kEventSingleThread` 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 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.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007)*

[SWS_CM_10307] Invoke the method - polling ⌈In case a `MethodCallProcessingMode` of `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 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.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007)*

[SWS_CM_10447]{DRAFT} **Dealing with unmodelled `ApApplicationError`s** ⌈If the service method (see [SWS_CM_00191]) returnes 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.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007)*

[SWS_CM_10308] Conditions for sending of a SOME/IP response message ⌈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 `ApApplicationError`s 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).⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007)*

[SWS_CM_10309] Transport protocol for sending of a SOME/IP response message ⌈The SOME/IP response message shall be transmitted using the transport protocol defined by the attribute `SomeipServiceInterfaceDeployment.methodDeployment.transportProtocol` in the Manifest.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00010)*

[SWS_CM_10310] Source of a SOME/IP response message ⌈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.⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00010)*

[SWS_CM_10311] Destination of a SOME/IP response message ⌈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.⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007)*

[SWS_CM_10312] Content of the SOME/IP response message ⌈The entries in the SOME/IP response message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00038]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Method ID (see [PRS_SOMEIP_00038]) 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 ApApplicationErrors 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 E_NOT_OK (0x01) in case the ClientServerOperation raised one of the possible ApApplicationErrors 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].

⌋(*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 SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10428] payload representing application error ⌈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 ara::core::ErrorCode::Value().

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

- sizeOfUnionLengthField/=32bit
- sizeOfUnionTypeSelectorField/=8bit
- sizeOfStructLengthField/=16bit

[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. – Please consult the release notes of the AUTOSAR classic platform regarding details about this incompatibility issue and how to create a project specific work-around.

- 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

⌋(*RS_SOMEIP_00014*)

[SWS_CM_10313] Checks for a received SOME/IP response message ⌈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_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Method ID (see [PRS_SOMEIP_00038]) matches the methodId attribute of one of the SomeipMethodDeployments 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).⌋(*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*)

[SWS_CM_10314] Identifying the right method ⌈Using the Service ID (see [PRS_SOMEIP_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element as well as the Method ID (see [PRS_SOMEIP_00038]) and the methodId attribute of the SomeipMethodDeployments of the SomeipServiceInterfaceDeployment, the right method shall be identified.⌋ *(RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00006, RS_SOMEIP_00007, RS_SOMEIP_00021)*

[SWS_CM_10315] Discarding orphaned responses ⌈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.⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_10357] Distinguishing errors from normal responses ⌈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) or an error response (Message Type set to ERROR (0x81) **or** Return Code set to a value different from 0x0)[2] w.r.t. the further processing according to [SWS_CM_10316], [SWS_CM_10358], [SWS_CM_10429], [SWS_CM_10430] and [SWS_CM_10317].⌋ *(RS_CM_00204, RS_SOMEIP_00008)*

[SWS_CM_10316] Deserializing the payload - normal response messages ⌈Based on the method determined according to [SWS_CM_10314] the Payload of the response message shall be deserialized according to the SOME/IP serialization rules. – Therefore the ArgumentDataPrototypes with direction set to inout and out shall be deserialized according to their order.⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00028)* The SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10442] Failures during deserialization of response messages ⌈In case of failures during deserialization of response messages, the ara::com implementation shall make the Future returned by the function call operator (operator()) of the respective Method class (see [SWS_CM_00196]) ready according to [SWS_CM_10440].⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00028)*

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

[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 ApApplicationErrors are raised.

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 error response message shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation), and the Future returned by the function call operator (operator()) of the respective Method class (see [SWS_CM_00196]) shall be made ready according to [SWS_CM_10440].

If this computed ApApplicationError.errorCode attribute value does match more than one of the ApApplicationError.errorCode attributes of all ApApplicationErrors referenced in role possibleApError or in role possibleApErrorSet.apApplicationError by the ClientServerOperation, the error response message shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation), and the Future returned by the function call operator (operator()) of the respective Method class (see [SWS_CM_00196]) shall be made ready according to [SWS_CM_10440].⌋*(RS_CM_00204, RS_SOMEIP_00008)*

Note: This is for backward compatibility to old servers 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) ⌈If the Return Code see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) contains a value equal to 0x01 (E_NOT_OK) then the corresponding ApApplicationError shall be identfied by deserializing the Payload of the message according to the error payload format described in [SWS_CM_10428].⌋ *(RS_CM_00204, RS_SOMEIP_00008)*

[SWS_CM_10430] Handling invalid messages with Message Type set to ERROR (0x81) ⌈If the Return Code see [PRS_SOMEIP_00058] and [PRS_SOMEIP_00191]) contains a value NOT equal to 0x01 or the value is equal to 0x01, but either the contained payload does NOT comply with [SWS_CM_10428] or the application error identified by the deserialized ApApplicationErrorDomain.value and ApApplicationError.errorCode is not referenced in role possibleApError or in role possibleApErrorSet.apApplicationError by the related ClientServerOperation, the error response message shall be discarded, the incident shall be logged (if logging is enabled for the ara::com implementation), and the Future returned by the function call operator (operator()) of the respective Method class (see [SWS_CM_00196]) shall be made ready according to [SWS_CM_10440].⌋*(RS_CM_00204, RS_SOMEIP_00008)*

[SWS_CM_10317] Making the Future ready ⌈In order to make the Future returned by the function call operator (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_00347]) operation of the `Promise` corresponding to this `Future` shall be invoked. This will unblock any blocking `get`, `wait`, `wait_for`, and `wait_until` calls that have been performed on this `Future`. – The `set_value` operation shall be invoked in case of a received normal response message using the deserialized 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.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007, RS_SOMEIP_00008)*

[SWS_CM_10318] Invoke the notification function ⌈If a notification function has been registered with the `Future`'s `then` method (see [SWS_CM_00197]), this notification function shall be invoked.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007)*

7.8.1.7 Handling Fields

[SWS_CM_10319] Conditions for sending of a SOME/IP event message ⌈The sending of a SOME/IP event message shall be requested by invoking the `Update` method of the respective `Field` class (see [SWS_CM_00119]) or if the `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). An active subscriber is an adaptive application that has invoked the `Subscribe` method of the respective `Field` class (see [SWS_CM_00120]) and has not canceled the subscription by invoking the `Unsubscribe` method of the respective `Field` class (see [SWS_CM_00120]) and where the subscription has not yet expired since the TTL contained in the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has been exceeded.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_SOMEIP_00018)*

[SWS_CM_10320] Transport protocol for sending of a SOME/IP event message ⌈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]).⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00010)*

[SWS_CM_10321] Source of a SOME/IP event message ⌈The source address and the source port of the SOME/IP event message shall set according to [SWS_CM_10289].⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042)*

[SWS_CM_10322] Destination of a SOME/IP event message ⌈The destination address and the destination port of the SOME/IP event message shall be set according to [SWS_CM_10290].⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042)*

[SWS_CM_10323] Content of the SOME/IP event message ⌈The entries in the SOME/IP event message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `fieldDeployment.notifier.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 information whether the Session Handling is activated or deactivated for an event can be derived from the `sessionHandling` attribute contained in the `ApSomeipTransformationProps` that is referenced by the `TransformationPropsToServiceInterfaceElementMapping` that in turn points to the event.
- 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.

⌋(*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 SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10324] Checks for a received SOME/IP event message ⌈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).⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_SOMEIP_00019*, *RS_SOMEIP_00022*, *RS_SOMEIP_00003*, *RS_SOMEIP_00004*, *RS_SOMEIP_00009*, *RS_SOMEIP_00014*)

[SWS_CM_10325] Identifying the right event ⌈Using the Service ID (see [PRS_SOMEIP_00040]) and the `serviceInterfaceId` attribute of the `SomeipServiceInterfaceDeployment` element as well as the Event ID (see [PRS_SOMEIP_00040]) and the `eventId` attribute of the `SomeipFieldDeployment`.`notifier`s of the `SomeipServiceInterfaceDeployment`, the right event shall be identified.⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00201*, *RS_SOMEIP_00004*, *RS_SOMEIP_00009*, *RS_SOMEIP_00022*)

[SWS_CM_10380] Silently discarding SOME/IP event messages for unsubscribed events ⌈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).⌋(*RS_CM_00204*, *RS_CM_00203*, *RS_SOMEIP_00004*, *RS_SOMEIP_00009*)

[SWS_CM_10328] Invoke receive handler ⌈In case a `ReceiveHandler` was registered using the `SetReceiveHandler` method (see [SWS_CM_00120]) of the respective `Field` class for the event determined according to [SWS_CM_10325] this registered receive handler shall be invoked.⌋(*RS_CM_00204*, *RS_CM_00203*, *RS_SOMEIP_00004*, *RS_SOMEIP_00009*)

[SWS_CM_10326] Deserializing the payload ⌈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 deserialized according to the SOME/IP serialization rules.⌋(*RS_CM_00204*, *RS_CM_00201*,

RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00028) The SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10327] Providing the received event data ⌈The deserialized payload containing the event data shall be provided via the `GetNewSamples` (see [SWS_CM_00701]) method of the respective `Field` class for the event determined according to [SWS_CM_10325].⌋*(RS_CM_00204, RS_CM_00202, RS_SOMEIP_00004, RS_SOMEIP_00009)*

[SWS_CM_10329] Conditions for sending of a SOME/IP request message ⌈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).⌋*(RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10443] Failures in sending of a SOME/IP request message ⌈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 `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].⌋*(RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10330] Transport protocol for sending of a SOME/IP request message ⌈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 protocol defined by the attribute `SomeipServiceInterfaceDeployment.fieldDeployment.get.transportProtocol` respectively.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010)*

[SWS_CM_10331] Source of a SOME/IP request message ⌈The source address and the source port of the SOME/IP request message shall be set according to [SWS_CM_10299].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010)*

[SWS_CM_10332] Destination of a SOME/IP request message ⌈The destination address and the destination port of the SOME/IP request message shall be set according to [SWS_CM_10300].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10333] Content of the SOME/IP request message ⌈The entries in the SOME/IP request message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00038]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Method ID (see [PRS_SOMEIP_00038]) 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.

⌋*(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 SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10334] Checks for a received SOME/IP request message ⌈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_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Method ID (see [PRS_SOMEIP_00038]) 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).⌋*(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)*

[SWS_CM_10335] Identifying the right method ⌈Using the Service ID (see [PRS_SOMEIP_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element as well as the Method ID (see [PRS_SOMEIP_00038]) and the methodId attribute of the SomeipFieldDeployment.sets and SomeipFieldDeployment.gets of the SomeipServiceInterfaceDeployment, the right method shall be identified.⌋*(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)*

[SWS_CM_10336] Deserializing the payload ⌈Based on the method determined according to [SWS_CM_10335] the Payload of the SOME/IP request message shall be deserialized according to the SOME/IP serialization rules.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00028)* The SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10338] Invoke the registered set/get handlers - event driven ⌈In case a MethodCallProcessingMode of either kEvent or kEventSingleThread 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 as a consequence to the reception of the SOME/IP request message.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10339] Invoke the registered set/get handlers - polling ⌈In case a `MethodCallProcessingMode` of `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.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10340] Conditions for sending of a SOME/IP response message ⌈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]).⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10341] Transport protocol for sending of a SOME/IP response message ⌈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.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010)*

[SWS_CM_10342] Source of a SOME/IP response message ⌈The source address and the source port of the SOME/IP response message shall be set according to [SWS_CM_10310].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00010)*

[SWS_CM_10343] Destination of a SOME/IP response message ⌈The destination address and the destination port of the SOME/IP response message shall be set according to [SWS_CM_10311].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10344] Content of the SOME/IP response message ⌈The entries in the SOME/IP response message shall be as follows:

- The Service ID (see [PRS_SOMEIP_00038]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Method ID (see [PRS_SOMEIP_00038]) 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 Future returned by the registered SetHandler resp. GetHandler or obtained internally) according to the SOME/IP serialization rules.

⌋(*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 SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10345] Checks for a received SOME/IP response message ⌈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).⌋(*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*)

[SWS_CM_10346] Identifying the right method ⌈Using the Service ID (see [PRS_SOMEIP_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element as well as the Method ID (see [PRS_SOMEIP_00038]) and the methodId attribute of the SomeipFieldDeployment.sets and SomeipFieldDeployment.gets of the SomeipServiceInterfaceDeployment,

the right method shall be identified.⌋*(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)*

[SWS_CM_10347] Discarding orphaned responses ⌈Orphaned responses shall be discarded according to [SWS_CM_10315].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_10348] Deserializing the payload ⌈Based on the method determined according to [SWS_CM_10346] the Payload of the SOME/IP response message shall be deserialized according to the SOME/IP serialization rules.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00028)* The SOME/IP serialization rules are explained in section 7.8.1.8.

[SWS_CM_10444] Failures during deserialization of response messages ⌈In case of failures during deserialization of response messages, the `ara::com` implementation shall make the `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].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_SOMEIP_00007, RS_SOMEIP_00009, RS_SOMEIP_00028)*

[SWS_CM_10349] Making the Future ready ⌈In order to make the `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 `Promise` corresponding to this `Future` shall be invoked using the deserialized payload as an argument. This will unblock any blocking `get`, `wait`, `wait_for`, and `wait_until` calls that have been performed on this `Future`.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10350] Invoke the notification function ⌈Any registered notification function shall be invoked according to [SWS_CM_10318].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215, RS_SOMEIP_00007, RS_SOMEIP_00009)*

[SWS_CM_10363]{DRAFT} **Failures in sending a SOME/IP event message** ⌈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]).⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_CM_00004)*

7.8.1.8 Serialization of Payload

[SWS_CM_10034] Serialization of Payload ⌈The serialization of the payload shall be based on the definition of the `ServiceInterface` of the data.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00028)*

[SWS_CM_10169] Missing parameters ⌈To allow migration the deserialization shall ignore parameters attached to the end of previously known parameter list.⌋*(RS_CM_00204, RS_CM_00202)*

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_10259] Seralization Padding ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)* This requirement does not apply for the serialization of extensible structs and methods (see chapter 7.8.1.8.4).

[SWS_CM_10260] Setting the alignment for a variable data length data element ⌈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.⌋*(RS_CM_00204, RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)* (see chapter 7.8.1.8.4)

[SWS_CM_11262] Missing alignment for a variable data length data element ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)* (see chapter 7.8.1.8.4)

[SWS_CM_11263] Precedence of alignment settings for a variable data length data element ⌈If `SomeipDataPrototypeTransformationProps.someipTransformationProps.alignment` and `TransformationPropsToServiceInterfaceElementMapping.transformationProps.alignment` are both not set for a variable data length data element, no alignment shall be applied.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10263] Padding for a fixed length data element ⌈After serialized fixed data length data elements, the SOME/IP network binding shall never add automatically a padding for alignment.⌋*(RS_CM_00201, RS_CM_00211)*

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 ⌈Alignment shall always be calculated from start of SOME/IP message.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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.

[SWS_CM_10016] Deserializing of exceeded unexpected data ⌈If more data than expected shall be deserialized, the unexpected data shall be discarded. The known fraction shall be considered.⌋*(RS_CM_00204, RS_CM_00202)*

[SWS_CM_10017] Deserializing incomplete data belonging to a field ⌈If less data than expected shall be deserialized and the data to be deserialized belong to a `Field`, the `initValue` should be used if it is defined. Otherwise the data shall be discarded and the incident shall be logged (if logging is enabled for the `ara::com` implementation).⌋*(RS_CM_00204, RS_CM_00202)*

In the following the serialization of different parameters is specified.

7.8.1.8.1 Basic Data Types

[SWS_CM_10036] Serialization of supported StdCppImplementationDataTypes ⌈The primitive `StdCppImplementationDataType`s defined in [13] which shall be supported for serialization are listed in Table 7.1.⌋*(RS_CM_00204, 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)

Table 7.1: Primitive `StdCppImplementationDataType`s supported for serialization

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

7.8.1.8.2 Enumeration Data Types

[SWS_CM_10361] Serializing Enumeration Data Type ⌈`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])⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.8.1.8.3 Scale Linear And Texttable Data Types

[SWS_CM_10391] Serializing Scale Linear And Texttable Data Type ⌈`Scale Linear And Texttable Data Type` shall be serialized according to [SWS_CM_10361] based on the `Enumeration Data Type` they were specified with (see [SWS_LBAP_00031]).⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.8.1.8.4 Structured Data Types (structs)

[SWS_CM_10042] Serializing a struct Data Type ⌈A `Structure Cpp Implementation Data Type` shall be serialized in order of depth-first traversal.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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 ⌈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 `ApSomeipTransformationProps` is

defined via SomeipDataPrototypeTransformationProps.someipTransformationProps.⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10252] ⌈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.⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10268] Setting the size length field for structs ⌈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.⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_00253] Default size of length field for structs ⌈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.⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_00254] Precedence when setting size of length field for structs ⌈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.⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_10269] Setting the byte order of the length field for structs ⌈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.⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*)

[SWS_CM_00255] Default size of length field for structs ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10270] Default byte order for the length field of structs ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10218] Scope of length field value for structs ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10219] Length greater than expected struct length ⌈If the length is greater than the expected length of a struct (as specified in the data type definition) a deserializing SOME/IP network binding shall only interpret the expected data and skip the unexpected.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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.

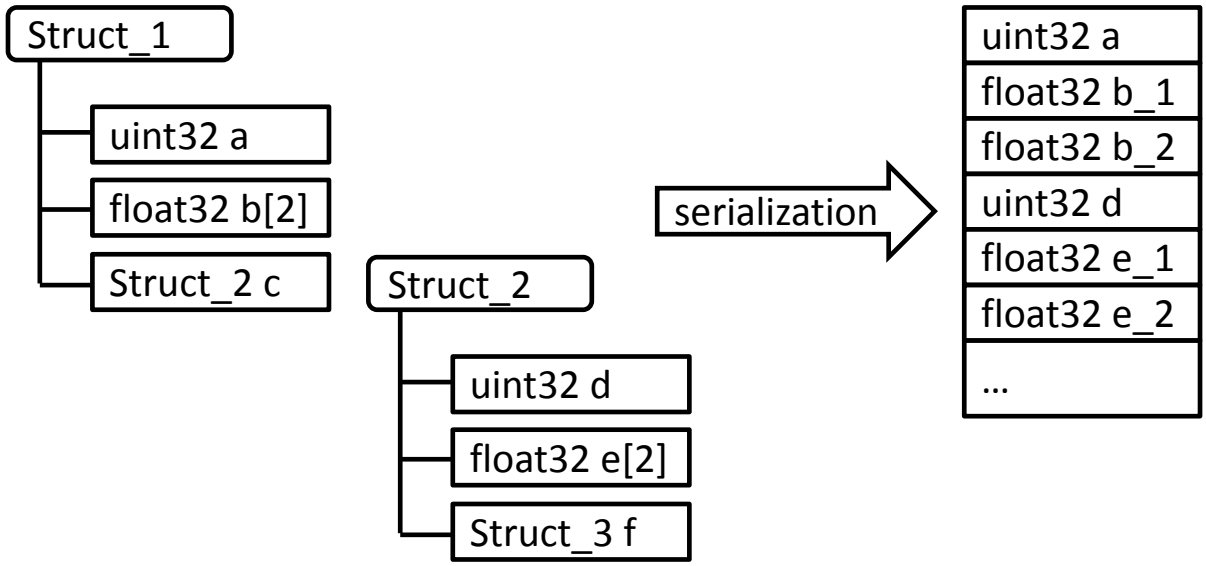


Figure 7.29: Serialization of Structs without Length Fields (Example)

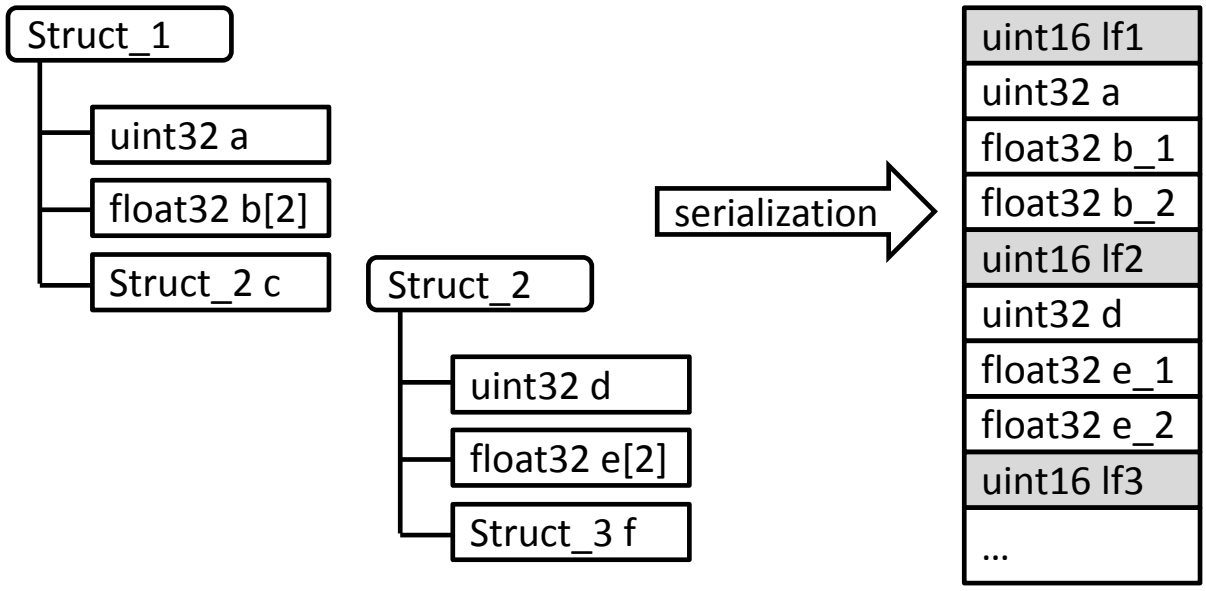


Figure 7.30: Serialization of Structs with Length Fields (Example)

[SWS_CM_01046] Definition of `tlvDataIdDefinition` ⌈Regarding the definition of `tlvDataIdDefinition` see [TPS_MANI_01097] and [constr_1594] for details.⌋ *(RS_CM_00204, RS_CM_00205, RS_SOMEIP_00050)*

7.8.1.8.5 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 [6] 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 8.1.2.5.2 Optional Data Types).

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 [5].

[SWS_CM_90443] Wire type for non-dynamic data types ⌈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`.⌋*(RS_CM_00204)*

[SWS_CM_90444] Wire type for dynamic data types ⌈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.⌋*(RS_CM_00204)*

[SWS_CM_90445] A deserializer shall always be able to handle the wire types 4, 5, 6 and 7 ⌈A deserializer shall always be able to handle the wire types 4, 5, 6 and 7 independent of the setting of `TransformationPropsToServiceInterfaceElementMapping.transformationProps.isDynamicLengthFieldSize`.⌋ *(RS_CM_00204)*

[SWS_CM_90446] Data ID ⌈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.⌋*(RS_CM_00204)*

Note: regarding existence of Data IDs, refer to [6].

Note: regarding existence of length field, refer to [5].

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 ⌈TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder shall define the byte order for the length field.⌋*(RS_CM_00204)*

[SWS_CM_90452] Default byte order for the length field of structs ⌈TransformationPropsToServiceInterfaceElementMapping.transformationProps.byteOrder is not defined, a byte order of mostSignificantByteFirst shall be used for the length field.⌋*(RS_CM_00204)*

Regarding structure members and serialization examples, refer to [5].

7.8.1.8.6 Strings

[SWS_CM_10053] Strings encoding ⌈Strings shall be encoded using Unicode and terminated with a "\0"-character.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10054] Supported encoding ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136)*

[SWS_CM_10285] Responsibility of proper string encoding ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136)*

[SWS_CM_10055] UTF-16LE and UTF-16BE terminating bytes ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10056] UTF-16LE and UTF-16BE strings length ⌈UTF-16LE and UTF-16BE strings shall have an even length.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10057] Odd UTF-16LE and UTF-16BE string length ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10248] Odd UTF-16LE and UTF-16BE string length ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10058] String start byte(BOM) ⌈All strings shall always start with a Byte Order Mark (BOM).⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

For the specification of BOM, see [14] and [15]. Please note that the BOM is used in the serialized strings to achieve compatibility with Unicode.

[SWS_CM_10459] Legacy string serialization ⌈The legacy string serialization shall be triggered if a Unicode is detected and attribute `ApSomeipTransformationProps.implementsLegacyStringSerialization` is true.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10059] BOM checking by SOME/IP network binding implementation ⌈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).⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10060] BOM addition ⌈The BOM shall be added by the SOME/IP sending network binding implementation.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10242] Model representation of UTF-8 Strings ⌈An UTF-8 String shall be represented by an `CppImplementationDataType`

- with `category` equal to `STRING`
- which may be mapped to an `ApplicationDataType` with `category` equal to `STRING` using a `DataTypeMap`
- with `ApplicationPrimitiveDataType.swDataDefProps.swTextProps.baseType.baseTypeDefinition.baseTypeEncoding` set to `UTF-8` in case that the `DataTypeMap` is defined.

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136)*

Please note that according to [constr_1674] the only supported encoding of `CppImplementationDataType` with `category` equal to `STRING` is UTF-8.

According to SOME/IP serialized strings start with a length field of 8, 16 or 32 bit which preceeds 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 ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10272] Byte order of length field for strings ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10273] Size of length field for strings ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10274] Setting byte order for the length field of strings ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10275] Default size of length field for strings ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10276] Default byte order for the length field of strings ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10277] Data type of the length field for strings ⌈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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10278] Data type of the length field for strings ⌈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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10245] Serialization of strings ⌈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).

⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*, *RS_AP_00136*)

[SWS_CM_10247]{DRAFT} **Deserialization of strings** ⌈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).
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.
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.
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).

⌋(*RS_CM_00204*, *RS_CM_00201*, *RS_CM_00202*, *RS_CM_00211*, *RS_AP_00136*)

7.8.1.8.7 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_00257] Missing size of array length field ⌈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]).⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10256] Size of the length field for arrays ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10279] Setting byte order for the length field of strings ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_00258] Default size of the length field for arrays ⌈If attribute TransformationPropsToServiceInterfaceElementMapping.transformationProps.sizeOfArrayLengthField is set to a value equal to 0 and attribute SomeipDataPrototypeTransformationProps.someipTransformationProps.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]).⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_00259] Setting size of the length field for arrays ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10280] Setting the byte order for size of length field for arrays ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10258] Default size of the length field for arrays ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10281] Byte order of length field for arrays ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10257] Datatype for the length field of arrays ⌈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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_00260] Datatype for the length field of arrays ⌈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

⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10076] Serializing arrays ⌈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⌋ *(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10234] Vector representation ⌈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 contained in the vector. Note that vectors are realized with dynamic sized arrays on SOME/IP level.⌋ *(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10235] Array representation ⌈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.⌋ *(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

In case of nested arrays, the same scheme applies.

[SWS_CM_10222] Setting the size of the length field for arrays ⌈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.⌋ *(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

The layout of arrays with dynamic length is shown in 7.31 and Figure 7.32 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.

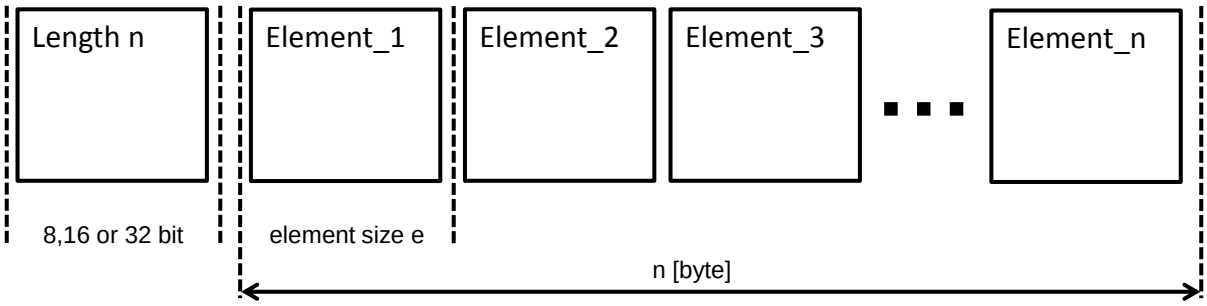


Figure 7.31: One-dimensional arrays (Example)

[SWS_CM_10070] Serializing one-dimentional array ⌈A one-dimensional array shall be serialized by concatenating the arrays elements in order.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

Multi-dimensional

[SWS_CM_10072] Serializing multi-dimentional array ⌈The serialization of multi-dimensional arrays shall happen in depth-first order.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

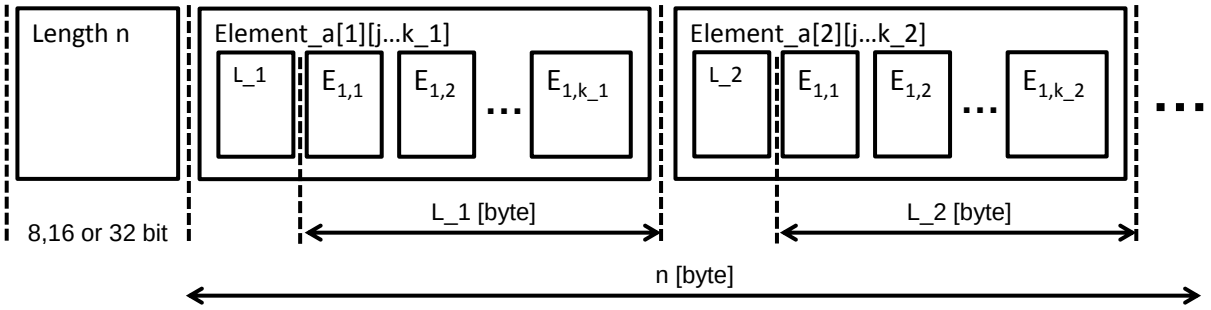


Figure 7.32: Multi-dimensional arrays (Example)

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 7.32.

7.8.1.8.8 Associative Maps

Associative map is modeled as `StdCppImplementationDataType` with `category` ASSOCIATIVE_MAP in the Manifest. As stated in the AUTOSAR Manifest Specification [6] 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 ⌈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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10262] Insertion of an associative map length field ⌈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]).⌋ *(RS_CM_00204, RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10282] Setting the byte order for size of the length field for associative maps ⌈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.⌋ *(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_00264] Setting the size of the length field for associative maps ⌈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]).⌋ *(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10283] Setting the byte order for size of the length field for associative maps ⌈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.⌋ *(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10267] Insertion of an associative map length field ⌈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.⌋ *(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10284] Default byte order for size of the length field for associative maps ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10264] Size of the associative map length field ⌈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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_00265] Datatype for the length field of associative maps ⌈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

⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10265] Serialization of associative map elements ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

Table 7.2 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 7.2: Example of a serialized associative map

[SWS_CM_10266] Applicability of mandatory padding after variable length data elements ⌈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.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)* (see chapter 7.8.1.8.4)

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 `ApplicationRecordElement`s.

7.8.1.8.9 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] Serialization layout of Variants ⌈The default serialization layout of Variants are specified by the union data type in SOME/IP which is shown in Table 7.3.⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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

Table 7.3: Default serialization layout of unions (Variants)

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 deserialized by the SOME/IP network binding.

[SWS_CM_10254] Variant length field ⌈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.⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10255] Variant length field data type ⌈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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10226] Serialized Variant size ⌈The serializing SOME/IP network binding shall write the size (in bytes) of the serialized Variant (including padding bytes but without the size of the length field and type field) into the length field of the Variant. This requirement does not apply for the serialization of extensible structs and methods.⌋ *(RS_CM_00201, RS_CM_00202, RS_CM_00211)* (see chapter 7.8.1.8.4)

[SWS_CM_10227] Length greater than expected Variant length ⌈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.⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

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.

The type field describes the type of the element. The length of the type field can be 32, 16, 8 or 0 bits.

[SWS_CM_10250] Data type for the length field of variants ⌈The data type of the type field of a Variant shall be determined using the `ara::core::Variant::index()` member function. The Variant template class is specified in [16].⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10251] Value of the variant type field ⌈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.⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10098] Possible values of the variant type field ⌈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.⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10099] Serialization of variant types ⌈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].⌋*(RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_10230]{DRAFT} **Data type for size of union field** ⌈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

⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.8.1.8.9.1 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

7.8.1.8.10 Segmentation of SOME/IP messages

[SWS_CM_10454] event message segmentation ⌈If the `SomeipEventDeployment` aggregates `SomeipEventDeployment.maximumSegmentLength` the `SOME/IP event message` shall be transmitted/received using segmentation as described in [PRS_SOMEIP_00720] and following. If a `SomeipEventDeployment.separationTime` is provided, it shall be considered on sender side.⌋*(RS_SOMEIP_00051)*

[SWS_CM_10455] method request message segmentation ⌈If the `SomeipMethodDeployment` aggregates `SomeipMethodDeployment.maximumSegmentLengthRequest` the `SOME/IP request message` shall be transmitted/received using segmentation as described in [PRS_SOMEIP_00720] and following. If a `SomeipMethodDeployment.separationTimeRequest` is provided, it shall be considered on sender side.⌋*(RS_SOMEIP_00051)*

[SWS_CM_10456] Message segmentation for the get and set methods of fields ⌈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.⌋*(RS_SOMEIP_00051)*

[SWS_CM_10457] Small messages segmentation ⌈For messages that would fit into one segment no segmentation (i.e. no `TP-Header`) shall be applied.⌋*(RS_SOMEIP_00051)*

[SWS_CM_10445]{DRAFT} **SomeIpBurstTransmission** ⌈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.⌋*(RS_SOMEIP_00051)*

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

7.8.1.9 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 ⌈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

⌋(*RS_CM_00101*)

7.8.2 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 two approaches how the signal-based information is made available at the adaptive AUTOSAR `Machine`:

- Network binding (see section 7.8.2.1)
- Network binding (see section 7.8.2.2)

7.8.2.1 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]{DRAFT} **Definition of serialization technology** ⌈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.⌋(*RS_CM_00204*)

See also chapter 7.8.2.1.8 and chapter 7.8.1.8.

In figure 7.33 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.

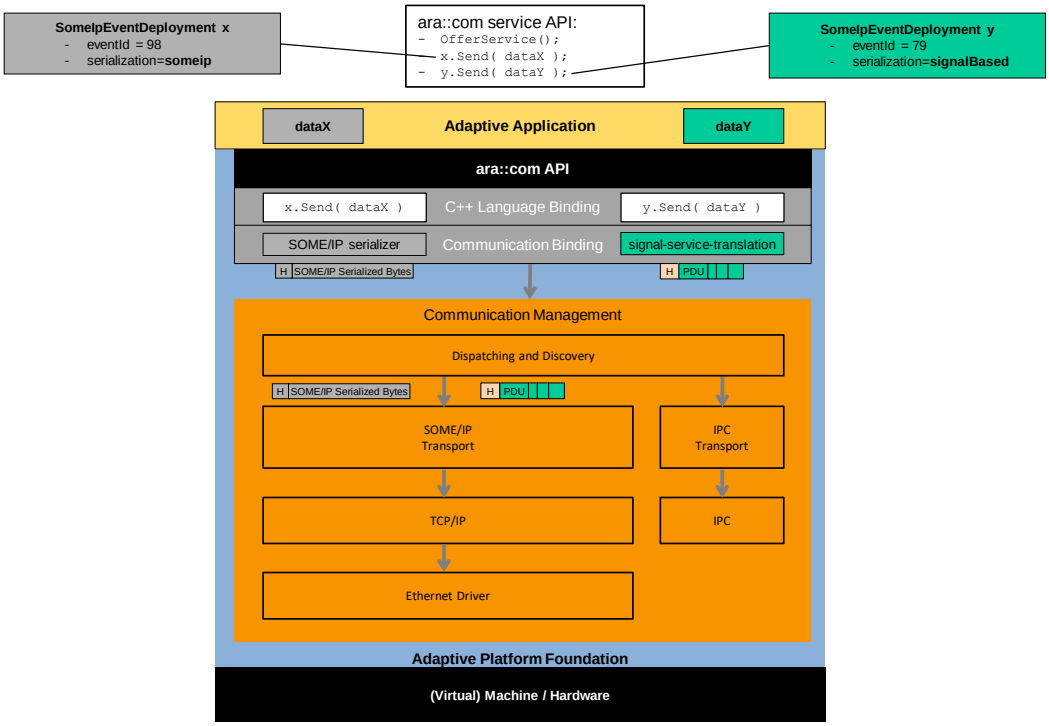


Figure 7.33: Example serialization settings

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 [6].

[SWS_CM_10174]{DRAFT} **Mix of signal-based and SOME/IP communication** ⌈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 [17] with enabled PDU-header option on the sender side.⌋(*RS_CM_00204*)

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]{DRAFT} ⌈The signal-based network binding shall implement the SOME/IP Service Discovery Protocol defined in [12] and the SOME/IP Protocol defined in [5] (except for the serialization of signal-based payload).⌋*(RS_CM_00204, RS_CM_00205, RS_CM_00004)*

[SWS_CM_10013] applies.

This means that Length and Type fields shall be always in network byte order.

Based on [SWS_CM_10172]:

[SWS_CM_80003]{DRAFT} **Byte order for signal-based network binding with SOME/IP serialization** ⌈If SomeipEventDeployment.serializer is set to someip then
the byte order of the parameters inside the payload shall be defined by byteOrder of ApSomeipTransformationProps.⌋*(RS_CM_00204, RS_SOMEIP_00026, RS_CM_00004)*

[SWS_CM_80004]{DRAFT} **Byte order for signal-based network binding with signal-based serialization** ⌈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.⌋*(RS_CM_00004)*

[SWS_CM_10240] applies.

7.8.2.1.1 Service Discovery

The section 7.8.1.1 is fully applicable to the signal-based network binding.

7.8.2.1.2 Accumulation of messages

Based on [SWS_CM_10387]:

[SWS_CM_80017]{DRAFT} **Data accumulation for UDP data transmission** ⌈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.⌋*(RS_CM_00204, RS_CM_00004)*

[SWS_CM_10388] applies.

Based on [SWS_CM_10389]:

[SWS_CM_80019]{DRAFT} **Configuration of a data accumulation on a `ProvidedSomeipServiceInstance` for transmission over UDP** ⌈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.

⌋*(RS_CM_00204, RS_CM_00004)*

Based on [SWS_CM_10390]:

[SWS_CM_80020]{DRAFT} **Configuration of a data accumulation on a `RequiredSomeipServiceInstance` for transmission over UDP** ⌈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.

⌋*(RS_CM_00204, RS_CM_00004)*

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].

7.8.2.1.3 Execution context of message reception actions

The section 7.8.1.3 is fully applicable to the signal-based network binding.

7.8.2.1.4 Handling Events

Based on [SWS_CM_10287]:

[SWS_CM_80021]{DRAFT} **Conditions for sending of an event message** ⌈The sending of an event message shall be requested by invoking the `Send` method of the respective `Event` class (see [SWS_CM_00162] and [SWS_CM_90437]) 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). An active subscriber is an adaptive application that has invoked the `Subscribe` method of the respective `Event` class (see [SWS_CM_00141]) and has not canceled the subscription by invoking the `Unsubscribe` method of the respective `Event` class (see [SWS_CM_00151]) and where the subscription has not yet expired since the TTL contained in the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has been exceeded.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_CM_00004)*

Based on [SWS_CM_10288]:

[SWS_CM_80022]{DRAFT} **Transport protocol for sending of an event message** ⌈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]).⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00010, RS_CM_00004)*

Based on [SWS_CM_10289]:

[SWS_CM_80023]{DRAFT} **Source of an event message** ⌈The event message shall use the unicast IP address and port taken from the IPv4/v6 Endpoint Option (see [PRS_SOMEIPSD_00304]) of the SOME/IP OfferService message

([SWS_CM_00203]) as source address and source port for the transmission.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042, RS_CM_00004)*

Based on [SWS_CM_10290]:

[SWS_CM_80024]{DRAFT} **Destination of an event message** ⌈The event message shall use the multicast IP address and the port taken from the IPv4/v6 Multicast Option (see [PRS_SOMEIPSD_00322]) 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_00304]) 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.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00042, RS_CM_00004)*

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]{DRAFT} **Content of the SOME/IP serialized event message** ⌈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_00040]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines 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_80240], 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_80240], 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 information whether the Session Handling is activated or deactivated for an event can be derived from the `sessionHandling` attribute contained in the `SomeipTransformationProps` that is referenced by the `TransformationPropsToServiceInterfaceElementMapping` that in turn points to the event.

- 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.

⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_CM_00004)*

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 `Event ID`) (see [SWS_CM_80026]).

[SWS_CM_80026]{DRAFT} **Content of the signal-based serialized event message** ⌈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_00040]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines 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 [6].

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00201*, *RS_SOMEIP_00041*, *RS_SOMEIP_00022*, *RS_SOMEIP_00003*, *RS_SOMEIP_00004*, *RS_CM_00004*)

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

Based on [SWS_CM_10292]:

[SWS_CM_80027]{DRAFT} **Checks for a received SOME/IP serialized event message** ⌈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_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Event ID (see [PRS_SOMEIP_00040]) matches 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 event message shall be discarded and the incident shall be logged (if logging is enabled for the ara::com implementation).⌋(*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_00004*)

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 Event ID) (see [SWS_CM_80028]).

[SWS_CM_80028]{DRAFT} **Checks for a received signal-based serialized event message** ⌈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_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Event ID (see [PRS_SOMEIP_00040]) matches the eventId attribute of one of the SomeipEventDeployments 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).⌋*(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_00004)*

[SWS_CM_10293] applies.

Based on [SWS_CM_10379]:

[SWS_CM_80030]{DRAFT} **Silently discarding event messages for unsubscribed events** ⌈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).⌋*(RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004, RS_CM_00004)*

[SWS_CM_10296] applies.

Based on [SWS_CM_10294]:

[SWS_CM_80032]{DRAFT} **Deserializing the SOME/IP serialized payload** ⌈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 deserialized according to the SOME/IP serialization rules.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00028, RS_CM_00004)*

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

[SWS_CM_80033]{DRAFT} **Deserializing the signal-based serialized payload** ⌈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 deserialized according to the signal-service-translation serialization rules defined in TPS-ManifestSpecification [6].⌋*(RS_CM_00004)*

[SWS_CM_10295] applies.

[SWS_CM_10360] applies.

7.8.2.1.5 Handling Triggers

[SWS_CM_10518]{DRAFT} **Conditions for sending of a trigger** ⌈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). An active subscriber is an adaptive application that has invoked the Subscribe method of the respective Trigger class (see [SWS_CM_00723]) and has not canceled the subscription by invoking the Unsubscribe method of the respective Trigger class (see [SWS_CM_00810]) and where the subscription has not yet expired since the TTL contained in the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has been exceeded.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_CM_00004)*

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 7.8.2.1.4 are also valid for Triggers 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]{DRAFT} **Content of the SOME/IP serialized trigger message** ⌈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_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines 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_80240], 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_80240], 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 information whether the Session Handling is activated or deactivated for a trigger can be derived from the sessionHandling attribute contained in the ApSomeipTransformationProps that is referenced by the TransformationPropsToServiceInterfaceElementMapping that in turn points to the trigger.
- 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).

⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00201, RS_SOMEIP_00041, RS_SOMEIP_00022, RS_SOMEIP_00003, RS_SOMEIP_00004, RS_CM_00004)*

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 Event ID) (see [SWS_CM_10520]).

[SWS_CM_10520]{DRAFT} **Content of the signal-based serialized trigger message** ⌈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_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the eventDeployment.eventId.
- The Length shall be set to 0.

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00201*, *RS_SOMEIP_00041*, *RS_SOMEIP_00022*, *RS_SOMEIP_00003*, *RS_SOMEIP_00004*, *RS_CM_00004*)

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

[SWS_CM_10521]{DRAFT} **Checks for a received SOME/IP serialized trigger message** ⌈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_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Event ID (see [PRS_SOMEIP_00040]) matches 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).⌋(*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_00004*)

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 Event ID) (see [SWS_CM_10520]).

[SWS_CM_10522]{DRAFT} **Checks for a received signal-based serialized trigger**
⌈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_00040]) and the serviceInterfaceId attribute of the SomeipServiceInterfaceDeployment element in the Manifest to determine the right ServiceInterface.
- Verify that the Event ID (see [PRS_SOMEIP_00040]) matches the eventId attribute of one of the SomeipEventDeployments 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).⌋*(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_00004)*

[SWS_CM_10514] applies.

[SWS_CM_10523]{DRAFT} **Silently discarding trigger for unsubscribed triggers**
⌈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_00249] shall *not* be performed).⌋*(RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004, RS_CM_00004)*

[SWS_CM_00249] applies.

7.8.2.1.6 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 7.8.1.6.

7.8.2.1.7 Handling Fields

Based on [SWS_CM_10319]:

[SWS_CM_80063]{DRAFT} **Conditions for sending of an event message** ⌈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 `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). An active subscriber is an adaptive application that has invoked the `Subscribe` method of the respective `Field` class (see [SWS_CM_00120]) and has not canceled the subscription by invoking the `Unsubscribe` method of the respective `Field` class (see [SWS_CM_00120]) and where the subscription has not yet expired since the TTL contained in the SOME/IP SubscribeEventgroup message (see [SWS_CM_00205]) has been exceeded.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00005, RS_SOMEIP_00017, RS_SOMEIP_00018, RS_CM_00004)*

Based on [SWS_CM_10320]:

[SWS_CM_80064]{DRAFT} **Transport protocol for sending of an event message** ⌈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]).⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00010, RS_CM_00004)*

Based on [SWS_CM_10321]:

[SWS_CM_80065]{DRAFT} **Source of an event message** ⌈The source address and the source port of the event message shall set according to [SWS_CM_80023].⌋ *(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042, RS_CM_00004)*

Based on [SWS_CM_10322]:

[SWS_CM_80066]{DRAFT} **Destination of an event message** ⌈The destination address and the destination port of the event message shall be set according to [SWS_CM_80024].⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00042, RS_CM_00004)*

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]{DRAFT} **Content of the SOME/IP serialized event message** ⌈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_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the serviceInterfaceId.
- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the SomeipServiceInterfaceDeployment element defines the fieldDeployment.notifier.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 information whether the Session Handling is activated or deactivated for an event can be derived from the sessionHandling attribute contained in the ApSomeipTransformationProps that is referenced by the TransformationPropsToServiceInterfaceElementMapping that in turn points to the event.
- 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.

⌋(*RS_CM_00204*, *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 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 `Event ID`) (see [SWS_CM_80068]).

[SWS_CM_80068]{DRAFT} **Content of the signal-based serialized event message** ⌈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_00040]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines the `serviceInterfaceId`.
- The Event ID (see [PRS_SOMEIP_00040]) shall be derived from the Manifest where the `SomeipServiceInterfaceDeployment` element defines 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 [6].

⌋(*RS_CM_00204*, *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 the `serializer` is defined as `someip serializer` the SOME/IP serialized event handling applies.

Based on [SWS_CM_10324]:

[SWS_CM_80069]{DRAFT} **Checks for a received SOME/IP serialized event message** ⌈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).⌋(*RS_CM_00204*, *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 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 Event ID) (see [SWS_CM_80070]).

[SWS_CM_80070]{DRAFT} **Checks for a received signal-based event message** ⌈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).⌋*(RS_CM_00004)*

[SWS_CM_10325] applies.

Based on [SWS_CM_10380]:

[SWS_CM_80072]{DRAFT} **Silently discarding event messages for unsubscribed events** ⌈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).⌋*(RS_CM_00204, RS_CM_00203, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_CM_00004)*

[SWS_CM_10328] applies.

Based on [SWS_CM_10326]:

[SWS_CM_80074]{DRAFT} **Deserializing the SOME/IP serialized payload** ⌈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 deserialized according to the SOME/IP serialization rules.⌋*(RS_CM_00204, RS_CM_00201, RS_SOMEIP_00004, RS_SOMEIP_00009, RS_SOMEIP_00028, RS_CM_00004)*

Note: [SWS_CM_80074] supports the mix of signal-based and SOME/IP communication use case defined in [SWS_CM_10174].

[SWS_CM_80075]{DRAFT} **Deserializing the signal-based payload** ⌈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 deserialized according to the signal-service-translation serialization rules defined in TPS-ManifestSpecification [6].⌋
(RS_CM_00004)

[SWS_CM_10327] applies.

[SWS_CM_10329] applies.

[SWS_CM_10443] applies.

[SWS_CM_10330] applies.

[SWS_CM_10331] applies.

[SWS_CM_10332] applies.

[SWS_CM_10333] applies.

[SWS_CM_10334] applies.

[SWS_CM_10335] applies.

[SWS_CM_10336] applies.

[SWS_CM_10338] applies.

[SWS_CM_10339] applies.

[SWS_CM_10340] applies.

[SWS_CM_10341] applies.

[SWS_CM_10342] applies.

[SWS_CM_10343] applies.

[SWS_CM_10344] applies.

[SWS_CM_10345] applies.

[SWS_CM_10346] applies.

[SWS_CM_10347] applies.

[SWS_CM_10348] applies.

[SWS_CM_10444] applies.

[SWS_CM_10349] applies.

[SWS_CM_10350] applies.

[SWS_CM_10363] applies.

7.8.2.1.8 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 7.8.1.8) is responsible for the handling of the serialization.

[SWS_CM_80100]{DRAFT} **SOME/IP serialization of signal-based network binding** ⌈If the attribute SomeipEventDeployment.serializer is set to someip then the serialization of the payload shall be based on the SOME/IP serialization rules.⌋ *(RS_CM_00004)*

Note: SOME/IP serialization rules are defined in section 7.8.1.8.

[SWS_CM_80101]{DRAFT} **ServiceInstanceToSignalMapping input for serialization of signal-based network binding** ⌈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 [6].⌋ *(RS_CM_00004)*

[SWS_CM_80102]{DRAFT} **Ignoring not mapped elements** ⌈To allow migration the deserialization shall ignore signals which are not subject to ServiceInstanceToSignalMapping.⌋ *(RS_CM_00004, RS_CM_00204, RS_CM_00202)*

[SWS_CM_80103]{DRAFT} **Deserializing incomplete data belonging to a field** ⌈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 (if logging is enabled for the ara::com implementation).⌋ *(RS_CM_00004, RS_CM_00204, RS_CM_00202)*

7.8.2.2 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.

7.8.2.2.1 Service Discovery

[SWS_CM_80501]{DRAFT} **Mapping of Offer Service (Signal-Based Static network binding)** ⌈When instructed to *offer* a service instance which is mapped to a ProvidedUserDefinedServiceInstance of category SIGNAL-BASED_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 localTcpPort-Number 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.⌋*(RS_CM_00004, RS_CM_00204)*

[SWS_CM_80512]{DRAFT} **Mapping of Stop Offer Service (Signal-Based Static network binding)** ⌈When instructed to *stop offering* a service instance which is mapped to a ProvidedUserDefinedServiceInstance of category SIG-NALBASED_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.

⌋*(RS_CM_00004, RS_CM_00204)*

[SWS_CM_80502]{DRAFT} **Mapping of Find Service (Signal-Based Static network binding)** ⌈When instructed to *find* a service instance which is mapped to a RequiredUserDefinedServiceInstance of category SIGNAL-BASED_WITH_HEADER or SIGNALBASED_NO_HEADER, then the Signal-Based Static network binding shall immediately return a ara::com::ServiceHandle-Container 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)

⌋*(RS_CM_00004, RS_CM_00204)*

[SWS_CM_80503]{DRAFT} **Mapping of Subscribe Service (Signal-Based Static network binding)** ⌈When instructed to *subscribe* to an event which is part of a RequiredUserDefinedServiceInstance of category SIGNAL-BASED_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.⌋*(RS_CM_00004, RS_CM_00204)*

[SWS_CM_80513]{DRAFT} **Mapping of Unsubscribe Service (Signal-Based Static network binding)** ⌈When instructed to *un-subscribe* from an event which is part of a RequiredUserDefinedServiceInstance of category SIGNAL-BASED_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.

⌋*(RS_CM_00004, RS_CM_00204)*

7.8.2.2.2 Accumulation of messages

[SWS_CM_80505]{DRAFT} **Data accumulation for UDP data transmission (Signal-Based Static network binding)** ⌈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.⌋ *(RS_CM_00004, RS_CM_00204)*

[SWS_CM_80504]{DRAFT} **Configuration of a data accumulation on a RequiredUserDefinedServiceInstance for transmission over UDP (Signal-Based Static network binding)** ⌈For a ProvidedUserDefinedServiceInstance of category SIGNALBASED_WITH_HEADER which has a udpCollectionBuffer-SizeThreshold > 0 defined, the events and field notifiers where udpCollection-Trigger 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.

⌋(*RS_CM_00004*, *RS_CM_00204*)

7.8.2.2.3 Execution context of message reception actions

The section 7.8.1.3 is fully applicable to the Signal-Based Static network binding.

7.8.2.2.4 Handling Events

[SWS_CM_80506]{DRAFT} **Arbitrary Message Header usage for Signal-Based Static network binding messages** ⌈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).⌋(*RS_CM_00004*, *RS_CM_00204*)

[SWS_CM_80507]{DRAFT} **No header option for Signal-Based Static network binding messages** ⌈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.⌋(*RS_CM_00004*, *RS_CM_00204*)

7.8.2.2.5 Handling Method Calls

[SWS_CM_80508]{DRAFT} **No method support for Signal-Based Static network binding** ⌈The Signal-Based Static network binding does not support methods.⌋(*RS_CM_00004*, *RS_CM_00204*)

7.8.2.2.6 Handling Fields

[SWS_CM_80509]{DRAFT} **Only field notifier support for `Signal-Based Static` network binding** ⌈The `Signal-Based Static` network binding only supports the field notifier. Getter or Setter methods are not supported.⌋*(RS_CM_00004, RS_CM_00204)*

7.8.2.2.7 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]{DRAFT} **Ignoring not mapped elements** ⌈To allow migration the deserialization shall ignore signals which are not subject to `ServiceInstanceToSignalMapping`.⌋*(RS_CM_00004)*

[SWS_CM_80511]{DRAFT} **Deserializing incomplete data belonging to a field** ⌈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 (if logging is enabled for the `ara::com` implementation).⌋*(RS_CM_00004)*

7.8.3 DDS Network binding

[SWS_CM_11000] DDS Compliance ⌈The DDS network binding shall comply with the DDS Minimum Profile defined in [18], the DDS Wire Interoperability protocol (RTPS) defined in [19], and the DDS-XTYPES Minimal Programming Interface and Network Interoperability Profiles defined in [20].⌋*(RS_CM_00204)*

[SWS_CM_90500]{DRAFT} **Choice of Service Instance discovery protocol** ⌈`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.⌋*(RS_CM_00101, RS_CM_00102)*

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 7.8.3.1 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 7.8.3.2 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]{DRAFT} **Topic naming for Domain Participant USER_DATA QoS - based Service Instances** ⌈When `DomainParticipantUserDataQos` is set in the `discoveryType` attribute for a specific `DdsProvidedServiceInstance` or `DdsRequiredServiceInstance`, the de-facto Topic naming scheme for events, methods and fields is the one described for `SERVICE_INSTANCE_RESOURCE_PARTITION`.⌋*(RS_CM_00201, RS_CM_00211, RS_CM_00216)*

7.8.3.1 Service Discovery via Domain Participant USER_DATA QoS policy

[SWS_CM_11001] Mapping of OfferService method ⌈When instructed to offer a Service, the DDS Binding shall perform the following operations:

- [SWS_CM_11002] It shall assign a DDS DomainParticipant to the Service Instance.
- [SWS_CM_11003] It shall assign a DDS Topic and a DDS DataWriter to every `VariableDataPrototype` defined in the `ServiceInterface` in the role `event`.
- [SWS_CM_10550] It shall assign a DDS Topic and a DDS DataWriter to every `Trigger` defined in the `ServiceInterface` in the role `trigger`.
- [SWS_CM_11029] 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_11030] 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_11031] 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_09004] It shall add the Service ID, Service Instance IDs, and `ServiceInterface` contract version to the DDS DomainParticipant's USER_DATA QoS Policy.

⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

[SWS_CM_11002] Assigning a DDS DomainParticipant to a Service Instance ⌈The DDS Binding shall assign a DDS DomainParticipant to every Service Instance. The configuration of the DomainParticipant is described in the TPS_ManifestSpecification:

- 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 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.

⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

[SWS_CM_11003] Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface ⌈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].

[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`).

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.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

[SWS_CM_10550]{DRAFT} **Assigning a DDS Topic and a DDS DataWriter to every Trigger in the ServiceInterface** ⌈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 triggers 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 triggers.

Topic objects may be reused across service instances; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

[SWS_CM_11029] Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Methods in the ServiceInterface ⌈The DDS binding shall instantiate a DDS Service [21] 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_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.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)* The handling of method calls with DDS is specified in 7.8.3.5.

[SWS_CM_11030] Assigning a DDS Topic and a DDS DataWriter to every Field in the ServiceInterface with its hasNotifier attribute equal to true ⌈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.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

[SWS_CM_11031] Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Field Getters/Setters in the ServiceInterface ⌈The DDS binding shall instantiate a DDS Service [21] 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.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)* The handling of fields with DDS is specified in section 7.8.3.6.

[SWS_CM_09004] Adding Service IDs, Service Instance IDs, and ServiceInterface Contract Versions to the DDS DomainParticipant's USER_DATA QoS Policy ⌈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, using the native DDS discovery mechanisms defined in [19]. The USER_DATA QoS Policy appends a user-defined value to the DomainParticipant's discovery messages. This information shall be used by ara::com Clients and DDS native applications to identify a DomainParticipant as an "ara::com DomainParticipant" that provides one or more Service Instances.

Service IDs, Service Instance IDs, and `ServiceInterface` contract versions shall be encoded in the USER_DATA QoS Policy in string format according to the following pattern:

```
"ara.com://services/<svcId>_<svcInId>-<svcMajVersion>.<svcMinVersion>
[&<svcId>_<svcInId>-<svcMajVersion>.<svcMinVersion>]*"
```

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`.

`<svcMajVersion>` is derived from the Manifest, where the `majorVersion` element of the `ServiceInterface` defines the contract's major version.

`<svcMinVersion>` is derived from the Manifest, where the `minorVersion` element of the `ServiceInterface` defines the contract's minor version.

Because a DomainParticipant may be associated with one or more Service Instances, the syntax specified above allows appending one or more `<svcId>_<svcInId>-<svcMajVersion>.<svcMinVersion>` pairs to the USER_DATA QoS:

- If USER_DATA QoS is empty, the binding implementation shall set it to `"ara.com://services/<svcId>_<svcInId>-<svcMajVersion>.<svcMinVersion>"`.
- Else, if USER_DATA QoS is not empty, the binding implementation shall append the Service ID and Instance to the current value preceded by an ampersand symbol (i.e., `"&<svcId>_<svcInId>-<svcMajVersion>.<svcMinVersion>"`).

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00101*, *RS_CM_00500*, *RS_CM_00501*)

[SWS_CM_11005] Mapping of StopOfferService method ⌈When instructed to stop offering a Service, the DDS Binding shall perform the following operations:

- It shall remove the appropriate Service and Instance IDs from the USER_DATA QoS Policy of the DDS DomainParticipant assigned to the Service Instance.
- 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 `field`s in the `ServiceInterface` with `hasGetter` and/or `hasSetter` attributes set to `true` created in previous calls to the `OfferService()` method.

⌋(*RS_CM_00204*, *RS_CM_00105*)

[SWS_CM_11006] Mapping of FindService method ⌈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 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.blacklistedVersion`.
- It shall return a `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.blacklistedVersion`. 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.

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00102*)

[SWS_CM_11007] Finding a DDS DomainParticipant suitable for performing client-side operations ⌈The DDS binding shall provide client-side methods with a DDS DomainParticipant capable of discovering and communicating with remote DDS DomainParticipants assigned to the requested Service Instance(s). The configuration of the DomainParticipant is described in the TPS_ManifestSpecification:

- The Domain ID of the DomainParticipant shall be derived from the Manifest, where the `DdsRequiredServiceInstance` element defines the `domainId`.
- The QoS Profile of the DomainParticipant shall be derived from the Manifest, where the `DdsRequiredServiceInstance` element defines the `qosProfile`.

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00102*)

[SWS_CM_11008] Creating a DDS DomainParticipant suitable for performing client-side operations ⌈To create a DomainParticipant capable of discovering and communicating with remote DDS DomainParticipants assigned to Service Instances, the binding implementation shall use the configuration parameters in the TPS_ManifestSpecification described in [SWS_CM_11007].⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00102*)

[SWS_CM_11009] Discovering remote Service Instances through DDS DomainParticipants ⌈DDS DomainParticipants created or retrieved in the context of Service Discovery are responsible for discovering remote DomainParticipants assigned to ara::com Service Instances.

To retrieve the list of discovered Service Instances, the DDS binding shall iterate first the list of remote DomainParticipants the DomainParticipant has discovered so far. This shall be done by calling `read()` on the DomainParticipant's built-in DataReader for the `DCPSParticipant` Topic. `DCPSParticipant` is a standard DDS Topic defined in [19] that DomainParticipants use to inform other DomainParticipants of their presence in the network. Among other things, `DCPSParticipant` Topics propagate the DomainParticipant's USER_DATA QoS Policy; therefore, these messages provide all the necessary information to identify remote DomainParticipants associated with ara::com Service Instances.

The DDS binding shall analyze the content of the USER_DATA QoS of each remote DomainParticipant and check whether they are associated with Service Instances matching the following criteria:

If `requiredServiceInstanceId` is set to "ALL", the binding shall return a new handle for each service instance found in remote DomainParticipants' USER_DATA QoS according to the following pattern:

```
"ara.com://services/.*<svcId>.*"
```

Else, if `requiredServiceInstanceId` is set to any value other than "ALL", the binding shall return a new handle for every service instance found in remote DomainParticipants' USER_DATA QoS according to the following pattern:

"ara.com://services/.*<svcId>_<reqSvcInId>.*"

Where:

<svcId> is the corresponding serviceInterfaceId.

<reqSvcInId> is the corresponding requiredServiceInstanceId.

In either case, before returning new handles the binding implementation shall evaluate the ServiceInterface contract version for the corresponding Service Instance in the content of the USER_DATA QoS. The binding shall return a new handle only if:

1. The ServiceInterface contract version of the discovered service instance is compatible with the serviceInterfaceDeployment version of the DdsRequiredServiceInstance according to [RS_CM_00501].
2. The ServiceInterface contract version is not part of any DdsRequiredServiceInstance.blacklistedVersion, according to [RS_CM_00701].

Before returning new handles, the binding implementation shall ensure that the DomainParticipant used in the discovery phase has one DDS Publisher and one DDS Subscriber per service instance found matching the filter criteria[5]. The Partition QoS of both DDS Publisher and DDS Subscriber shall contain the following partition name to match the partition in which the DataReaders and DataWriters associated with the remote service instance are operating (in consonance with [SWS_CM_11002]):

"ara.com://services/<svcId>_<reqSvcInId>"

If the binding implementation does not find a DDS Publisher with the aforementioned requirements, it shall create a new one and configure the Publisher's Partition QoS with the partition name defined above. Likewise, if it does not find a DDS Subscriber with those requirements, it shall create a new one and configure it accordingly.

Publisher and Subscriber objects may be reused across proxies associated with a remote service instance; therefore, they shall not be removed until the enclosing DomainParticipant is destroyed.

⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00102)*

[SWS_CM_11010] Mapping of StartFindService method ⌈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 FindServiceHandler upon the occurrence of any of the following events:

[5]These Publishers and Subscribers will be used to enclose all the DDS DataWriters and DataReaders, respectively, that will handle communication with the corresponding remote service instance's DDS DataReaders and DataWriters.

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.

⌋(*RS_CM_00204, RS_CM_00200, RS_CM_00102*)

[SWS_CM_11011] Defining a DDS BuiltinParticipantListener ⌈The DDS Binding implementation shall define a `BuiltinParticipantListener` class to handle notifications whenever a remote DomainParticipant is discovered. This class shall derive from the standard `DataReaderListener` class [18], specifying that the data type of the samples to be handled is `ParticipantBuiltinTopicData`—the data type associated with the built-in DataReader for samples of `DCPSParticipant` Topic [19].

`BuiltinParticipantListener` shall implement the following methods according to the specified instructions:

- A Constructor that takes as a parameter references to a `FindServiceHandler` and a `requiredServiceInstanceId`. These references shall be stored in member variables so that they can be used by subsequent executions of `on_data_available()`—which is the method the listener calls every time a new DomainParticipant is discovered.
- An `on_data_available()` method that calls `FindServiceHandler` using the value of the member variable `requiredServiceInstanceId`. If the returned `ServiceHandleContainer` contains more than one element, `on_data_available()` shall invoke `FindServiceHandler` and pass the container as a parameter; otherwise the method shall return and perform no further action.

⌋(*RS_CM_00204, RS_CM_00200, RS_CM_00102*)

[SWS_CM_11012] Binding a BuiltinParticipantListener to a DDS DomainParticipant ⌈To bind a `BuiltinParticipantListener` to a DDS DomainParticipant, the DDS binding implementation shall create a new `BuiltinParticipantListener` object (see [SWS_CM_11011]) passing `FindServiceHandler` and `requiredServiceInstanceId` to the listener's constructor. Then service shall then bind the newly created listener to the DomainParticipant using the `set_listener()` method with `StatusMask = DATA_AVAILABLE_STATUS`[6].

[6]Note that the syntax of `set_listener()` and `StatusMask` is described in terms of the DDS Platform-Independent Model specified in [18]. Different Platform-Specific Mappings, such as the DDS-CPP-PSM specified in [22], map these concepts into more language-friendly constructs.

The BuiltinParticipantListener shall be removed when the enclosing DomainParticipant is destroyed.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00102)*

[SWS_CM_11013] Mapping of StopFindService method ⌈When instructed to stop a continuous service search initiated by a previous call to `StartFindService()`, 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, `StopFindService()` shall return and perform no further action.
- [SWS_CM_11014] It shall unbind the `BuiltinParticipantListener` from the retrieved DDS DomainParticipant[7].

⌋*(RS_CM_00204, RS_CM_00200)*

[SWS_CM_11014] Unbinding a BuiltinParticipantListener from a DDS DomainParticipant ⌈When instructed to unbind a `BuiltinParticipantListener` from a DDS DomainParticipant, the DDS binding implementation service shall invoke the DomainParticipant's `set_listener()` method to disable the listener. In that case, `set_listener()` shall be called with `StatusMask = STATUS_MASK_NONE`.⌋*(RS_CM_00204, RS_CM_00200)*

7.8.3.2 Service Discovery via Topic

[SWS_CM_90502]{DRAFT} **Mapping of OfferService method** ⌈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

[7] Note that with the behavior specified for `FindService()` and `StartFindService()`—the only methods capable of creating DomainParticipants—guarantees that the DomainParticipant used by subsequent calls to `StartFindService()` and `StopFindService()` will be the same.

access to all the Fields 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

⌋(*RS_CM_00204, RS_CM_00200, RS_CM_00101*)

[SWS_CM_90503]{DRAFT} **Assigning a DDS DomainParticipant to a Service Instance** ⌈The DDS Binding shall assign a DDS DomainParticipant to every Service Instance. The configuration of the DomainParticipant is described in the TPS_ManifestSpecification:

- 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[8]. If the search is successful, the binding shall assign the DomainParticipant found to the Service[9]; 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 and DataReaders associated with the Service Instance.

⌋(*RS_CM_00204, RS_CM_00200, RS_CM_00101*)

[SWS_CM_90504]{DRAFT} **Assigning a DDS Topic and a DDS DataWriter to every Event in the ServiceInterface** ⌈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

[8]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.

[9]The rules specified in this binding ensure the creation of only one DomainParticipant for a given Domain and set of QoS settings (qosProfile).

[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.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

[SWS_CM_90505]{DRAFT} **Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Methods in the ServiceInterface** ⌈The DDS binding shall instantiate a DDS Service [21] 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.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

The handling of method calls with DDS is specified in 7.8.3.5.

[SWS_CM_90506]{DRAFT} **Assigning a DDS Topic and a DDS DataWriter to every Field in the ServiceInterface with its hasNotifier attribute equal to true** ⌈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.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

[SWS_CM_90507]{DRAFT} **Assigning a DDS Request and Reply Topic, and DataWriters and DataReaders, to the Field Getters/Setters in the ServiceInterface** ⌈The DDS binding shall instantiate a DDS Service [21] 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_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.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101)*

The handling of fields with DDS is specified in section 7.8.3.6.

[SWS_CM_90508]{DRAFT} **Advertising Service IDs, Service Instance IDs, and ServiceInterface Contract Versions over the `ara.com://services/discovery` topic** ⌈The binding implementation shall configure DDS Topic, Publisher and DataWriter objects supporting the publication of announcement messages over the `ara.com://services/discovery` topic, whose type is `ServiceAnnouncementMessage` and is defined as follows[10]:

```
module dds {
module ara {
module com {
```

[10]DDS types are often defined in OMG IDL [23], which provides a standard language-independent format to represent data types and interfaces. Even though we use IDL throughout the specification to define data types, the use of IDL to is not mandated (i.e., a compliant implementation could choose to hand-craft these types, run code generation from an equivalent XML syntax, or run vendor-specific mechanisms to generate the actual data types).

```
4
5  enum ServiceInstanceResourceIdentifierType {
6          SERVICE_INSTANCE_RESOURCE_PARTITION,
7          SERVICE_INSTANCE_RESOURCE_TOPIC_PREFIX,
8          SERVICE_INSTANCE_RESOURCE_INSTANCE_ID
9  };
10
11 struct ServiceVersion {
12         uint32 major_version;
13         uint32 minor_version;
14 };
15
16 struct ServiceAnnouncementMessage {
17         @key string<256> interface_id;
18         @key uint16 instance_id;
19         ServiceVersion version;
20         ServiceInstanceResourceIdentifierType identifier_type;
21 };
22
23 }; // module com
24 }; // module ara
25 }; // module dds
```

Where:

interface_id is the Service Instance ID derived from the Manifest, where the DdsServiceInterfaceDeployment defines the serviceInterfaceId. The value of this field contributes to the topic instance key

instance_id is the Service Instance ID derived from the Manifest, where the DdsProvidedServiceInstance element defines the serviceInterfaceId. The value of this field contributes to the topic instance key

version is derived from the Manifest, where the majorVersion element of the ServiceInterface defines the contract major version, and the minorVersion element of the ServiceInterface defines the contract minor version

identifier_type 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 requirements: [SWS_CM_11015], [SWS_CM_11100], [SWS_CM_11130], [SWS_CM_11144] and [SWS_CM_10524].

In order to guarantee reception of ServiceAnnouncementMessage samples by all Service Interface consumers, including those joining after the Service Instance has been advertised, the following DataWriter QoS policies shall be set for the ara.com://services/discovery topic:

- RELIABILITY set to RELIABLE
- HISTORY set to KEEP_LAST with DEPTH set to 1
- DURABILITY set to TRANSIENT_LOCAL

Once the `ara.com://services/discovery` topic DataWriter is properly set up and ready to use, the offering Service Instance shall:

1. Instantiate a `ServiceAnnouncementMessage` sample, update it with the proper values uniquely identifying the Service Instance, and use it to register via `register_instance()` a unique instance (keyed by `interface_id` and `instance_id`)
2. Use the Instance Handle returned by the previous step to publish the sample via `write()`
3. Keep a copy the sample and the Instance Handle for use upon Service Instance tear down (see [SWS_CM_11005])

⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00101, RS_CM_00500, RS_CM_00501)*

[SWS_CM_90509]{DRAFT} **Mapping of StopOfferService method** ⌈When instructed to stop offering a Service, the DDS Binding shall perform the following operations:

- Call `dispose()` using the sample and the Instance Handle kept during Service Instance announcement (see [SWS_CM_90508])
- 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.

⌋*(RS_CM_00204, RS_CM_00105)*

[SWS_CM_90510]{DRAFT} **Mapping of FindService method** ⌈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 a DataReader matching the Topic and QoS policies defined by [SWS_CM_90508], looking into all samples received 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.blacklistedVersion.

- It shall return a 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.blacklistedVersion.

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00102*)

[SWS_CM_90511]{DRAFT} **Finding a DDS DomainParticipant suitable for performing client-side operations** ⌈The DDS binding shall provide client-side methods with a DDS DomainParticipant capable of discovering and communicating with remote DDS DomainParticipants assigned to the requested Service Instance(s). The configuration of the DomainParticipant is described in the TPS_ManifestSpecification:

- The Domain ID of the DomainParticipant shall be derived from the Manifest, where the DdsRequiredServiceInstance element defines the domainId.
- The QoS Profile of the DomainParticipant shall be derived from the Manifest, where the DdsRequiredServiceInstance element defines the qosProfile.

⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00102*)

[SWS_CM_90512]{DRAFT} **Creating a DDS DomainParticipant suitable for performing client-side operations** ⌈To create a DomainParticipant capable of discovering and communicating with remote DDS DomainParticipants assigned to Service Instances, the binding implementation shall use the configuration parameters in the TPS_ManifestSpecification described in [SWS_CM_90511].⌋(*RS_CM_00204*, *RS_CM_00200*, *RS_CM_00102*)

[SWS_CM_90513]{DRAFT} **Discovering remote Service Instances through the ara.com://services/discovery topic** ⌈DDS DomainParticipants created or retrieved in the context of Service Discoverty are responsible for discovering remote DomainParticipants assigned to ara::com Service Instances.

To retrieve a list of discovered Service Instances, the DDS binding shall process inbound ServiceAnnouncementMessage samples from the ara.com://services/discovery topic. This shall be done by calling read() on the DataReader object defined by [SWS_CM_90510].

If requiredServiceInstanceId is set to ALL, the binding shall return a new handle for each service instance declared by inbound ServiceAnnouncementMessage, as long as its interface_id field matches the corresponding serviceInterfaceId.

Else, if requiredServiceInstanceId is set to any value other than ALL, the binding should return a new handle for each service instance declared by inbound ServiceAnnouncementMessage, as long as its interface_id field matches the serviceInterfaceId and its instance_id field matches requiredServiceInstanceId.

In either case, before returning new handles, the binding implementation shall evaluate the ServiceInterface contract version for the corresponding Service Instance in the content of the ServiceAnnouncementMessage samples. The binding shall return a new handle only if:

1. The ServiceInterface contract version of the discovered service instance is compatible with the serviceInterfaceDeployment version of the DdsRequiredServiceInstance according to [RS_CM_00501]
2. The ServiceInterface contract version is not part of any DdsRequiredServiceInstance.blacklistedVersion, according to [RS_CM_00701].

⌋(*RS_CM_00204, RS_CM_00200, RS_CM_00102*)

[SWS_CM_90514]{DRAFT} **Mapping of StartFindService method** ⌈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, calling FindServiceHandler whenever a matching Service Instance is discovered.

⌋(*RS_CM_00204, RS_CM_00200, RS_CM_00102*)

[SWS_CM_90515]{DRAFT} **Mapping of StopFindService method** ⌈When instructed to stop a continuous service search initiated by a previous call to StartFindService(), 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, StopFindService() shall return and perform no further action.
- It shall stop monitoring the arrival of ServiceAnnouncementMessage samples through the ara.com://services/discovery topic.

⌋(*RS_CM_00204, RS_CM_00200*)

7.8.3.3 Handling Events

[SWS_CM_11015] Mapping Events to DDS Topics ⌈The DDS binding shall map every VariableDataPrototype defined in the ServiceInterface in the role event to a DDS Topic. The equivalent DDS Topic shall be configured as follows:

- The Topic Name shall be derived from the Manifest according to the following rules:

  - If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION` or `SERVICE_INSTANCE_RESOURCE_INSTANCE_ID`, then the topic name shall be set to `ara.com://services/<InterfaceID>/<Major>.<Minor>/<TopicName>`
  - Additionally, if the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, then samples of this topic shall be sent and received via DataWriters and DataReaders whose respective parent Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: `ara.com://services/<InterfaceID>/<InstanceId>`
  - Finally, if the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_TOPIC_PREFIX`, then the topic name shall be set to `ara.com://services/<InterfaceID>/<InstanceID>/<TopicName>`
  - Where:

    <InterfaceID> is the value of `DdsServiceInterfaceDeployment.serviceInterfaceId`

    <InstanceID> is the value of either `DdsProvidedServiceInstance.serviceInstanceId` or `DdsRequiredServiceInstance.requiredServiceInstanceId`

    <Major> and <Minor> are the values of `ServiceInterface.majorVersion` and `ServiceInterface.minorVersion`, respectively

    <TopicName> is the value of `DdsEventDeployment.topicName`

- The Topic Data Type shall be defined as specified in [SWS_CM_11016], and shall be registered under the equivalent data type name.

⌋(*RS_CM_00204*, *RS_CM_00201*)

[SWS_CM_11016] DDS Topic data type definition ⌈The data type of a DDS Topic representing an Event shall be constructed according to the following IDL definition:

```
struct <eventTypeName>EventType {
    @key uint16 instance_id;
    <eventTypeName> data;
};
```

Where:

<eventTypeName> is the `Cpp Implementation Data Type symbol`

instance_id is a `@key` member of the type, which identifies all samples with the same `instance_id` as samples of the same Topic Instance.

data is the actual value of the `event`, which shall be constructed and encoded according to the DDS serialization rules. The `@external` annotation is optionally allowed, for cases where references yield implementation benefits over values.

⌋(*RS_CM_00204, RS_CM_00201*)

The DDS serialization rules are defined in section 7.8.3.7.

[SWS_CM_11017] Mapping of Send method ⌈When instructed to send an event message, the DDS Binding shall construct a new sample of the equivalent DDS Topic data type (see [SWS_CM_11016]) as follows:

- The Instance Id field (`instance_id`) shall be derived from the Manifest, where the `DdsProvidedServiceInstance` element defines the `serviceInstanceId`.
- The Data field (`data`) shall point to the `data` input parameter of the `Send()` method.

That sample shall be then passed as a parameter to the `write()` method of the DDS DataWriter associated with the `event`, which shall serialize the sample according to the serialization rules, and publish it over DDS.⌋(*RS_CM_00204, RS_CM_00201*)

The DDS serialization rules are defined in section 7.8.3.7.

[SWS_CM_11018] Mapping of Subscribe method ⌈When instructed to subscribe to an event, the DDS binding shall create a DDS DataReader using the DDS Subscriber created for the proxy in [SWS_CM_11009]. The rules to create the DataReader are specified in [SWS_CM_11019].

⌋(*RS_CM_00204, RS_CM_00103*)

[SWS_CM_11019] Creating a DDS DataReader for event subscription ⌈The DDS binding shall create a DDS DataReader for the Topic associated with the `event` (see [SWS_CM_11015]). If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the proxy communicates only with the service instance it is bound to, the binding implementation shall use the DDS Subscriber created in [SWS_CM_11002] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) to create the DataReader.

The DataReader shall be configured as follows:

- `DataReaderQos` shall be set as specified in the Manifest, where the `DdsEventQosProps` element defines the `qosProfile` that shall be used. To configure the DataReader's cache size according to the `maxSampleCount` specified in the `Subscribe()` method call, the value of the DataReader's HISTORY QoS specified in `qosProfile` shall be overridden as follows:

  - `history.kind = KEEP_LAST_HISTORY_QOS`
  - `history.depth = <maxSampleCount>`

- `Listener` shall be an instance of the `DataReaderListener` class specified in [SWS_CM_11020].
- `StatusMask` shall be set to `STATUS_MASK_NONE`.

⌋*(RS_CM_00204, RS_CM_00103)*

[SWS_CM_11020] Defining a DDS DataReaderListener ⌈The DDS Binding implementation shall define a `DataReaderListener` class capable of handling notifications when a new sample is received and/or when the matched status of the subscription changes. This class shall derive from the standard `DataReaderListener` class [18], specifying that the samples to be handled are of the Topic data type specified in [SWS_CM_11016].

The `DataReaderListener` shall implement the following methods according to the specified instructions:

- A Constructor that initializes two member variables that hold references to an `EventReceiveHandler` and a `SubscriptionStateChangeHandler`.
- An `on_data_available()` method that calls the `EventReceiveHandler` if it has been set and there are valid samples in the DataReader's cache.
- An `on_subscription_matched()` method that calls `GetSubscriptionState()` and passes the resulting `SubscriptionState` to `SubscriptionStateChangeHandler` if it has been set.
- A `set_event_receive_handler()` method that takes as an input parameter a reference to an `EventReceiveHandler` and updates the member variable holding a reference to an `EventReceiveHandler` to point to the input parameter.
- A `set_subscription_state_change_handler()` method that takes as an input parameter a reference to a `SubscriptionStateChangeHandler` and updates the member variable holding a reference to a `SubscriptionStateChangeHandler` to point to the input parameter.

⌋*(RS_CM_00204, RS_CM_00103)*

[SWS_CM_11021] Mapping of Unsubscribe method ⌈When instructed to unsubscribe from a service event, the DDS binding shall delete the DataReader associated with the `event`.⌋*(RS_CM_00204, RS_CM_00104)*

[SWS_CM_11022] Mapping of GetSubscriptionState method ⌈When instructed to provide the subscription state, the DDS binding shall check if the DataReader associated with the subscription exists:

- If it does exist, the binding shall call the DataReader's `get_subscription_matched_status()` method next.

  - If the `total_count` attribute of the resulting `SubscriptionMatchedStatus` is greater than zero, `GetSubscriptionState()` shall return `SubscriptionState = kSubscribed`.
  - Otherwise, it shall return `SubscriptionState = kSubscriptionPending`.
- Else, if it does not exist—which indicates that either `Subscribe()` has never invoked or `Unsubscribe()` has been called before—`GetSubscriptionState()` shall return `SubscriptionState = kNotSubscribed`.

⌋(*RS_CM_00204*, *RS_CM_00106*)

[SWS_CM_11023] Mapping of GetNewSamples method ⌈When instructed to get new samples, the DDS binding shall perform a `take()` on the DataReader as follows:

- 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.

⌋(*RS_CM_00204*, *RS_CM_00202*)

[SWS_CM_11024] Mapping of GetFreeSampleCount method ⌈When instructed to provide the number of free sample slots, the binding implementation shall return the number free sample slots in the DDS DataReader's cache.⌋(*RS_CM_00204*, *RS_CM_00202*)

[SWS_CM_11025] Mapping of SetReceiveHandler method ⌈When instructed to register an `EventReceiveHandler`, the binding implementation shall perform the following operations:

- It shall get a reference to the DataReader's listener using the `get_listener()` method.
- It shall use the `set_event_receive_handler()` method to instruct the listener to invoke the new `EventReceiveHandler` whenever there is data available.
- It shall update the DataReader's listener by calling `set_listener()` with `listener` equal to the new listener object and `StatusMask` set as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `DATA_AVAILABLE_STATUS`, set it to `DATA_AVAILABLE_STATUS`.

  - If the original value of `StatusMask` was `SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.

⌋(*RS_CM_00204*, *RS_CM_00203*)

[SWS_CM_11026] Mapping of UnsetReceiveHandler method ⌈When instructed to unregister an `EventReceiveHandler`, the binding implementation shall perform the following operations:

- It shall get a reference to the DataReader's listener using the `get_listener()` method.
- It shall use the `set_event_receive_handler()` method to unset the internal `EventReceiveHandler` that is called whenever there is data available.
- It shall update the DataReader's listener by calling `set_listener()` with `listener` equal to the new listener object and `StatusMask` set as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `DATA_AVAILABLE_STATUS`, set it to `STATUS_MASK_NONE`.
  - If the original value of `StatusMask` was `SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.

⌋(*RS_CM_00204*, *RS_CM_00203*)

[SWS_CM_11027] Mapping of SetSubscriptionStateHandler method ⌈When instructed to register a `SubscriptionStateChangeHandler`, the binding implementation shall perform the following operations:

- It shall get a reference to the DataReader's listener using the `get_listener()` method.
- It shall use the `set_subscription_state_change_handler()` method to instruct the listener to invoke the new `SubscriptionStateChangeHandler` whenever there is a change in the `SubscriptionMatchedStatus`.
- It shall update the DataReader's listener by calling `set_listener()` with `listener` equal to the new listener object and `StatusMask` set as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.

- If the original value of StatusMask was DATA_AVAILABLE_STATUS, set it to DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS.
- If the original value of StatusMask was DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS, set it to DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS.

⌋(*RS_CM_00204*, *RS_CM_00106*)

[SWS_CM_11028] Mapping of UnsetSubscriptionStateHandler method ⌈When instructed to unregister a SubscriptionStateChangeHandler, the binding implementation shall perform the following operations:

- It shall get a reference to the DataReader's listener using the get_listener() method.
- It shall use the set_subscription_state_change_handler() method to instruct the listener to unset the internal SubscriptionStateChangeHandler that is called whenever there is a change in the SubscriptionMatchedStatus.
- It shall update the DataReader's listener by calling set_listener() with listener equal to the new listener object and StatusMask set as follows:
  - If the original value of StatusMask was STATUS_MASK_NONE or SUBSCRIPTION_MATCHED_STATUS, set it to STATUS_MASK_NONE.
  - If the original value of StatusMask was DATA_AVAILABLE_STATUS, set it to DATA_AVAILABLE_STATUS.
  - If the original value of StatusMask was DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS, set it to DATA_AVAILABLE_STATUS.

⌋(*RS_CM_00204*, *RS_CM_00106*)

7.8.3.4 Handling Triggers

[SWS_CM_10524]{DRAFT} **Mapping Triggers to DDS Topics** ⌈The DDS binding shall map every Trigger defined in the ServiceInterface in the role trigger to a DDS Topic. The equivalent DDS Topic shall be configured as follows:

- The Topic Name shall be derived from the Manifest according to the following rules:
  - If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION or SERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the topic name shall

be set to ara.com://services/<InterfaceID>/<Major>.<Minor>/<TopicName>

- Additionally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this topic shall be sent and received via DataWriters and DataReaders whose respective parent Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: ara.com://services/<InterfaceID>/<InstanceId>
- Finally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_TOPIC_PREFIX, then the topic name shall be set to ara.com://services/<InterfaceID>/<InstanceID>/<TopicName>
- Where:

  <InterfaceID> is the value of DdsServiceInterfaceDeployment.serviceInterfaceId

  <InstanceID> is the value of either DdsProvidedServiceInstance.serviceInstanceId or DdsRequiredServiceInstance.requiredServiceInstanceId

  <Major> and <Minor> are the values of ServiceInterface.majorVersion and ServiceInterface.minorVersion, respectively

  <TopicName> is the value of DdsEventDeployment.topicName associated with the trigger

- The Topic Data Type shall be defined as specified in [SWS_CM_10525], and shall be registered under the equivalent data type name.

⌋(*RS_CM_00204, RS_CM_00201*)

[SWS_CM_10525]{DRAFT} **DDS Topic data type definition** ⌈The data type of a DDS Topic representing a trigger shall be constructed according to the following IDL definition:

```
struct TriggerType {
    @key uint16 instanceIdentifier;
};
```

Where:

instance_id is a @key member of the type, which identifies all samples with the same instance_id as samples of the same Topic Instance.

⌋(*RS_CM_00204, RS_CM_00201*)

[SWS_CM_10526]{DRAFT} **Mapping of Send method** ⌈When instructed to send a trigger message, the DDS Binding shall construct a new sample of the equivalent DDS Topic data type (see [SWS_CM_10525]) as follows:

- The Instance Id field (`instance_id`) shall be derived from the Manifest, where the `DdsProvidedServiceInstance` element defines the `serviceInstanceId`.

That sample shall be then passed as a parameter to the `write()` method of the DDS DataWriter associated with the `trigger`, which shall serialize the sample according to the serialization rules, and publish it over DDS.⌋*(RS_CM_00204, RS_CM_00201)*

The DDS serialization rules are defined in section 7.8.3.7.

[SWS_CM_10527]{DRAFT} **Mapping of Subscribe method** ⌈When instructed to subscribe to a trigger, the DDS binding shall create a DDS DataReader using the DDS Subscriber created for the proxy in [SWS_CM_11009] or [SWS_CM_90513]. The rules to create the DataReader are specified in [SWS_CM_10528].⌋*(RS_CM_00204, RS_CM_00103)*

[SWS_CM_10528]{DRAFT} **Creating a DDS DataReader for trigger subscription** ⌈The DDS binding shall create a DDS DataReader for the Topic associated with the `trigger` (see [SWS_CM_10524]). If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the proxy communicates only with the service instance it is bound to, the binding implementation shall use the DDS Subscriber created in [SWS_CM_11009] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) to create the DataReader.

The DataReader shall be configured as follows:

- `DataReaderQos` shall be set as specified in the Manifest, where the `DdsEventQosProps` element defines the `qosProfile` that shall be used.
- `Listener` shall be an instance of the `DataReaderListener` class specified in [SWS_CM_11020].
- `StatusMask` shall be set to `STATUS_MASK_NONE`.

⌋*(RS_CM_00204, RS_CM_00103)*

[SWS_CM_10529]{DRAFT} **Defining a DDS DataReaderListener** ⌈The DDS Binding implementation shall define a `DataReaderListener` class capable of handling notifications when a new sample is received and/or when the matched status of the subscription changes. This class shall derive from the standard `DataReaderListener` class [18], specifying that the samples to be handled are of the Topic data type specified in [SWS_CM_10525].

The `DataReaderListener` shall implement the following methods according to the specified instructions:

- A Constructor that initializes two member variables that hold references to an `TriggerReceiveHandler` and a `SubscriptionStateChangeHandler`.
- An `on_data_available()` method that calls the `TriggerReceiveHandler` if it has been set and there are valid samples in the DataReader's cache.
- An `on_subscription_matched()` method that calls `GetSubscriptionState()` and passes the resulting `SubscriptionState` to `SubscriptionStateChangeHandler` if it has been set.
- A `set_trigger_receive_handler()` method that takes as an input parameter a reference to an `TriggerReceiveHandler` and updates the member variable holding a reference to an `TriggerReceiveHandler` to point to the input parameter.
- A `set_subscription_state_change_handler()` method that takes as an input parameter a reference to a `SubscriptionStateChangeHandler` and updates the member variable holding a reference to a `SubscriptionStateChangeHandler` to point to the input parameter.

⌋(*RS_CM_00204, RS_CM_00103*)

[SWS_CM_10530]{DRAFT} **Mapping of Unsubscribe method** ⌈When instructed to unsubscribe from a service trigger, the DDS binding shall delete the DataReader associated with the `trigger`.⌋(*RS_CM_00204, RS_CM_00104*)

[SWS_CM_10531]{DRAFT} **Mapping of GetSubscriptionState method** ⌈When instructed to provide the subscription state, the DDS binding shall check if the DataReader associated with the subscription exists:

- If it does exist, the binding shall call the DataReader's `get_subscription_matched_status()` method next.
  - If the `total_count` attribute of the resulting `SubscriptionMatchedStatus` is greater than zero, `GetSubscriptionState()` shall return `SubscriptionState = kSubscribed`.
  - Otherwise, it shall return `SubscriptionState = kSubscriptionPending`.
- Else, if it does not exist—which indicates that either `Subscribe()` has never invoked or `Unsubscribe()` has been called before—`GetSubscriptionState()` shall return `SubscriptionState = kNotSubscribed`.

⌋(*RS_CM_00204, RS_CM_00106*)

[SWS_CM_10532]{DRAFT} **Mapping of GetNewTriggers method** ⌈When instructed to get new triggers, the DDS binding shall perform a `take()` on the DataReader without specifying a `max_samples` limit.

After calling `take()`, the binding implementation shall increase the internal trigger count proportionally to the number of samples returned by take().⌋*(RS_CM_00204, RS_CM_00202)*

[SWS_CM_10534]{DRAFT} **Mapping of SetReceiveHandler method** ⌈When instructed to register an `TriggerReceiveHandler`, the binding implementation shall perform the following operations:

- It shall get a reference to the DataReader's listener using the `get_listener()` method.
- It shall use the `set_trigger_receive_handler()` method to instruct the listener to invoke the new `TriggerReceiveHandler` whenever there is data available.
- It shall update the DataReader's listener by calling `set_listener()` with `listener` equal to the new listener object and `StatusMask` set as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `DATA_AVAILABLE_STATUS`, set it to `DATA_AVAILABLE_STATUS`.
  - If the original value of `StatusMask` was `SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.

⌋*(RS_CM_00204, RS_CM_00203)*

[SWS_CM_10535]{DRAFT} **Mapping of UnsetReceiveHandler method** ⌈When instructed to unregister an `TriggerReceiveHandler`, the binding implementation shall perform the following operations:

- It shall get a reference to the DataReader's listener using the `get_listener()` method.
- It shall use the `set_trigger_receive_handler()` method to unset the internal `TriggerReceiveHandler` that is called whenever there is data available.
- It shall update the DataReader's listener by calling `set_listener()` with `listener` equal to the new listener object and `StatusMask` set as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `DATA_AVAILABLE_STATUS`, set it to `STATUS_MASK_NONE`.
  - If the original value of `StatusMask` was `SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.

⌋(*RS_CM_00204*, *RS_CM_00203*)

[SWS_CM_10536]{DRAFT} **Mapping of SetSubscriptionStateHandler method** ⌈When instructed to register a `SubscriptionStateChangeHandler`, the binding implementation shall perform the following operations:

- It shall get a reference to the DataReader's listener using the `get_listener()` method.
- It shall use the `set_subscription_state_change_handler()` method to instruct the listener to invoke the new `SubscriptionStateChangeHandler` whenever there is a change in the `SubscriptionMatchedStatus`.
- It shall update the DataReader's listener by calling `set_listener()` with `listener` equal to the new listener object and `StatusMask` set as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `SUBSCRIPTION_MATCHED_STATUS`, set it to `SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`.

⌋(*RS_CM_00204*, *RS_CM_00106*)

[SWS_CM_10537]{DRAFT} **Mapping of UnsetSubscriptionStateHandler method** ⌈When instructed to unregister a `SubscriptionStateChangeHandler`, the binding implementation shall perform the following operations:

- It shall get a reference to the DataReader's listener using the `get_listener()` method.
- It shall use the `set_subscription_state_change_handler()` method to instruct the listener to unset the internal `SubscriptionStateChangeHandler` that is called whenever there is a change in the `SubscriptionMatchedStatus`.
- It shall update the DataReader's listener by calling `set_listener()` with `listener` equal to the new listener object and `StatusMask` set as follows:
  - If the original value of `StatusMask` was `STATUS_MASK_NONE` or `SUBSCRIPTION_MATCHED_STATUS`, set it to `STATUS_MASK_NONE`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS`, set it to `DATA_AVAILABLE_STATUS`.
  - If the original value of `StatusMask` was `DATA_AVAILABLE_STATUS|SUBSCRIPTION_MATCHED_STATUS`, set it to `DATA_AVAILABLE_STATUS`.

⌋(*RS_CM_00204*, *RS_CM_00106*)

7.8.3.5 Handling Method Calls

The RPC over DDS Specification (DDS-RPC) [21] 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 [18]. 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 ⌈Every `ServiceInterface` containing one or more `ClientServerOperation`s defined in the role `method` shall have an associated DDS Service to enable ara::com Service Instances to offer those operations, and to enable client applications to invoke them. The equivalent DDS Service shall provide all of the `method`s of the corresponding `ServiceInterface`.

DDS Services shall be constructed according to the Basic Service Mapping Profile of the RPC over DDS specification [21], which assigns two DDS Topics to every DDS Service: a Request Topic and a Reply Topic. Thus, every `ServiceInterface` containing one or more `ClientServerOperation`s defined in the role `method` shall trigger the creation of two equivalent DDS Topics.

The equivalent DDS Request Topic shall be configured as follows:

- The Request Topic Name shall be derived from the Manifest according to the following rules:
  - If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION` or `SERVICE_INSTANCE_RESOURCE_INSTANCE_ID`, then the topic name shall be set to `ara.com://services/<InterfaceID>/<Major>.<Minor>/<TopicName>`
  - Additionally, if the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, then samples of this topic shall be sent and received via DataWriters and DataReaders whose respective parent Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: `ara.com://services/<InterfaceID>/<InstanceId>`
  - Finally, if the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_TOPIC_PREFIX`, then the topic name shall be set

to ara.com://services/<InterfaceID>/<InstanceID>/<TopicName>

<InterfaceID> is the value of DdsServiceInterfaceDeployment.serviceInterfaceId

<InstanceID> is the value of either DdsProvidedServiceInstance.serviceInstanceId or DdsRequiredServiceInstance.requiredServiceInstanceId

<Major> and <Minor> are the values of ServiceInterface.majorVersion and ServiceInterface.minorVersion, respectively

<TopicName> is the value of DdsServiceInterfaceDeployment.methodRequestTopicName

- The Request Topic Data Type shall be defined as specified in [SWS_CM_11101], and shall be registered under the equivalent data type's name.

The equivalent DDS Reply Topic shall be configured as follows:

- The Reply Topic Name shall be derived from the Manifest according to the following rules:
  - If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION or SERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the topic name shall be set to ara.com://services/<InterfaceID>/<Major>.<Minor>/<TopicName>
  - Additionally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this topic shall be sent and received via DataWriters and DataReaders whose respective parent Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: ara.com://services/<InterfaceID>/<InstanceId>
  - Finally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_TOPIC_PREFIX, then the topic name shall be set to ara.com://services/<InterfaceID>/<InstanceID>/<TopicName>
  - Where:

    <InterfaceID> is the value of DdsServiceInterfaceDeployment.serviceInterfaceId

<InstanceID> is the value of either DdsProvidedServiceInstance.serviceInstanceId or DdsRequiredServiceInstance.requiredServiceInstanceId

<Major> and <Minor> are the values of ServiceInterface.majorVersion and ServiceInterface.minorVersion, respectively

<TopicName> is the value of DdsServiceInterfaceDeployment.methodReplyTopicName

- The Reply Topic Data Type shall be defined as specified in [SWS_CM_11102], and shall be registered under the equivalent data type's name.

⌋(*RS_CM_00204, RS_CM_00212, RS_CM_00213*)

[SWS_CM_11101] DDS Service Request Topic data type definition ⌈As specified in section 7.5.1.1.6 of [21], the Request Topic data type is a structure composed of a Request Header with meta-data a Call Structure with data. The IDL definition of the Request Topic data type is the following:

```
struct <svcId>Method_Request {
    dds::rpc::RequestHeader header;
    <svcId>Method_Call data;
};
```

Where:

<svcId> is the corresponding serviceInterfaceId.

dds::rpc::RequestHeader is the standard Request Header defined in section 7.5.1.1.1 of [21].

<svcId>Method_Call is the union that holds the value of the input parameters of the corresponding methods, according to the rules specified in section 7.5.1.1.6 of [21].

dds::rpc::RequestHeader shall be constructed as specified in section 7.5.1.1.1 of [21]. On top of that, the binding implementation shall set instanceName (a member of the RequestHeader structure that specifies the DDS Service instance name) to a string representation of the serviceInstanceId of the service instance that provides the methods.

<svcId>Method_Call shall be constructed as specified in section 7.5.1.1.6 of [21]:

- The name of the union shall be <svcId>Method_Call.
- The union discriminator shall be a 32-bit signed integer.
- The union shall have a default case of type dds::rpc::UnknownOperation (defined in section 7.5.1.1.1 of [21]) for unsupported and unknown operations.
- The union shall have a case label for each ClientServerOperation defined in the ServiceInterface with the role method, where:

- The integer value of the case label shall be a 32-bit hash of the ClientServerOperation's shortName. The binding implementation shall compute the hash as specified in section 7.5.1.1.2 of [21]. Representations of the service interface in OMG IDL [23] shall define 32-bit signed integer constants (i.e., const int32 <svcId>Method_<methodName>_Hash; where <methodName> is the shortName of the ClientServerOperation) to simplify the representation of the union cases (see below).
- The member name for the case label shall be the shortName of the ClientServerOperation.
- The type for each case label shall be <svcId>Method_<methodName>_In, which shall be constructed as specified in section 7.5.1.1.4 of [21] (see below).

The IDL definition of the <svcId>Method_Call union is the following:

```
union <svcId>Method_Call switch(int32) {
default:
    dds::rpc::UnknownOperation unknownOp;
case <svcId>Method_<method0Name>_Hash:
    <svcId>Method_<method0Name>_In <method0Name>;
case <svcId>Method_<method1Name>_Hash:
    <svcId>Method_<method1Name>_In <method1Name>;
// ...
case <svcId>Method_<methodNName>_Hash:
    <svcId>Method_<methodNName>_In <methodNName>;
};
```

As defined in section 7.5.1.1.4 of [21], the <svcId>Method_<methodName>_In structure shall contain as members all the ArgumentDataPrototypes of the ClientServerOperation with direction set to in or inout. The IDL representation of <svcId>Method_<methodName>_In is the following:

```
struct <svcId>Method_<methodName>_In {
    <ArgumentDataPrototype[0]>;
    <ArgumentDataPrototype[1]>;
    // ...
    <ArgumentDataPrototype[n]>;
};
```

In accordance with [21], for methods with no input parameters, the DDS binding shall generate a <svcId>Method_<methodName>_In structure with a single member named dummy of type dds::rpc::UnusedMember (see section 7.5.1.1.1 of [21]).

The resulting Request Topic data type shall be encoded according to the DDS serialization rules. Unions, such as the <svcId>Method_Call union, shall be serialized as specified in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00200)*

[SWS_CM_11102] DDS Service Reply Topic data type definition ⌈As specified in section 7.5.1.1.7 of [21], the Reply Topic data type is a structure composed of a Reply

Header with meta-data and a Return Structure with data. The IDL definition of the Reply Topic data type is the following:

```
struct <svcId>Method_Reply {
    dds::rpc::ReplyHeader header;
    <svcId>Method_Return data;
};
```

Where:

<svcId> is the corresponding serviceInterfaceId.

dds::rpc::ReplyHeader is the standard Reply Header defined in section 7.5.1.1.1 of [21].

<svcId>Method_Return is the union that holds the return values (i.e., return values, output parameter values, and/or errors) of the corresponding response, according to the rules specified in section 7.5.1.1.7 of [21].

dds::rpc::ReplyHeader shall be constructed as specified in section 7.5.1.1.1 of [21].

<svcId>Method_Return shall be constructed as specified in section 7.5.1.1.7 of [21]:

- The name of the union shall be <svcId>Method_Return.
- The union discriminator shall be a 32-bit signed integer.
- The union shall have a default case of type dds::rpc::UnknownOperation (defined in section 7.5.1.1.1 of [21]) for unsupported and unknown operations.
- The union shall have a case label for each ClientServerOperation defined in the ServiceInterface with the role method, where:
  - The integer value of the case label shall be a 32-bit hash of the ClientServerOperation's shortName. The binding implementation shall compute the hash as specified in section 7.5.1.1.2 of [21]. Representations of the service interface in OMG IDL [23] shall define 32-bit signed integer constants (i.e., const int32 <svcId>Method_<methodName>_Hash; where <methodName> is the shortName of the ClientServerOperation) to simplify the representation of the union cases (see below).
  - The member name for the case label shall be the shortName of the ClientServerOperation.
  - The type for each case label shall be <svcId>Method_<methodName>_Result, which shall be constructed as specified in section 7.5.1.1.4 of [21] (see below).

The IDL definition of <svcId>Method_Return is the following:

```
union <svcId>Method_Return switch(int32) {
default:
    dds::rpc::UnknownOperation unknownOp;
```

```
case <svcId>Method_<method0Name>_Hash:
    <svcId>Method_<method0Name>_Result <method0Name>;
case <svcId>Method_<method1Name>_Hash:
    <svcId>Method_<method1Name>_Result <method1Name>;
// ...
case <svcId>Method_<methodNName>_Hash:
    <svcId>Method_<methodNName>_Result <methodNName>
};
```

As defined in section 7.5.1.1.5 of [21], the `<svcId>Method_<methodName>_Result` union shall be constructed as follows:

- The union discriminator shall be a 32-bit signed integer.
- The union shall have a case with label `dds::RETCODE_OK` to represent a successful return:
  - The value of `RETCODE_OK` shall be 0x00, as specified in section 2.3.3 of [18].
  - The successful case shall have a single member named `result` of type `<svcId>Method_<methodName>_Out` (see below).
- The union shall also have a case with label `dds::RETCODE_ERROR` to represent the `ApApplicationError` the method may return:
  - The value of `RETCODE_ERROR` shall be 0x01, as specified in section 2.3.3 of [18].
  - The error case shall have a single member named `error` of type `ara::core::ErrorCode` (see [SWS_CM_10428]).

The IDL representation of `<svcId>Method_<methodName>_Result` is the following:

```
union <svcId>Method_<methodName>_Result switch(int32) {
case dds::RETCODE_OK:
    <svcId>Method_<methodName>_Out result;
case dds::RETCODE_ERROR:
    ara::core::ErrorCode error;
};
```

Lastly, as defined in section 7.5.1.1.5 of [21], the `<svcId>Method_<methodName>_Out` structure be constructed as follows:

- The structure shall contain as members all the `ArgumentDataPrototype`s of the `ClientServerOperation` with `direction` set to `out` or `inout`.
- The members of the structure representing `out` and `inout` arguments shall appear in the structure in the same order as they were declared.
- If the method has no `out`, and no `inout` arguments, the structure shall contain a single member named `dummy` of type `dds::rpc::UnusedMember` (in accordance with section 7.5.1.1.1 of [21]).

The IDL representation of <svcId>Method_<methodName>_Out is the following:

```
struct <svcId>Method_<methodName>_Out {
    <ArgumentDataPrototype[0]>;
    <ArgumentDataPrototype[1]>;
    // ...
    <ArgumentDataPrototype[n]>;
};
```

The resulting Reply Topic data type shall be encoded according to the DDS serialization rules. Unions, such as the <svcId>Method_<methodName>_Result union, shall be serialized as specified in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00200)*

[SWS_CM_10431] Mapping of ara::core::ErrorCode ⌈A ApApplicationError shall be represented according to the following IDL [23]:

```
module dds {
module ara {
module core {

struct ErrorCode {
    uint64 error_domain_value;
    int32 error_code;
};

}; // module core
}; // module ara
}; // module dds
```

Where:

error_domain_value is a 64-bit unsigned integer representing the ApApplicationErrorDomain. value, to which the raised ApApplicationError belongs.

error_code is a 32-bit signed integer representing the ApApplicationError. errorCode, which is represented on binding level as ara::core::ErrorCode::Value().

ara::core::ErrorCode shall be serialized according to the DDS serialization rules. 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.⌋*(RS_CM_00204)*

The DDS serialization rules are defined in section 7.8.3.7.

[SWS_CM_11103] Creating a DataWriter to handle method requests on the client side ⌈The DDS binding shall create a DDS DataWriter for the Request Topic associated with the methods of the ServiceInterface (see [SWS_CM_11101]) upon proxy instantiation.

If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, to ensure

the proxy communicates only with the service instance it is bound to, the binding implementation shall use the DDS Publisher created in [SWS_CM_11009] (whose partition name is "ara.com://services/<svcId>_<reqSvcInId>") to create the DataWriter.

The DataWriter shall be configured as follows:

- DataWriterQos shall be set as specified in the Manifest, where the DdsRequiredServiceInstance element defines the qosProfile that shall be used.

⌋(*RS_CM_00204*, *RS_CM_00212*, *RS_CM_00213*)

[SWS_CM_11104] Creating a DataReader to handle method responses on the client side ⌈The DDS binding shall create a DDS DataReader for the Reply Topic associated with the methods of the ServiceInterface (see [SWS_CM_11102]) upon proxy instantiation.

If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, to ensure the proxy communicates only with the service instance it is bound to, the binding implementation shall use the DDS Subscriber created in [SWS_CM_11009] (whose partition name is "ara.com://services/<svcId>_<reqSvcInId>") to create the DataReader.

The DataReader shall be configured as follows:

- DataReaderQos shall be set as specified in the Manifest, where the DdsRequiredServiceInstance element defines the qosProfile that shall be used.

⌋(*RS_CM_00204*, *RS_CM_00212*, *RS_CM_00213*, *RS_CM_00215*)

[SWS_CM_11105] Creating a DataReader to handle method requests on the server side ⌈The DDS binding shall create a DDS DataReader for the Request Topic associated with the methods of the ServiceInterface (see [SWS_CM_11101]) as part of the OfferService() operation (see [SWS_CM_11001]).

If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, the binding shall use the DDS Subscriber created in [SWS_CM_11002] (whose partition name is "ara.com://services/<svcId>_<svcInId>") to create the DataReader.

The DataReader shall be configured as follows:

- DataReaderQos shall be set as specified in the Manifest, where the DdsProvidedServiceInstance element defines the qosProfile that shall be used.
- Listener and StatusMask shall be set according to the value of MethodCallProcessingMode that was selected in the constructor of the ServiceSkeleton class:

  - For `MethodCallProcessingMode = kEvent` or `kEventSingleThread`, `Listener` shall be set to an instance of the `DataReaderListener` class specified in [SWS_CM_11110], and `StatusMask` shall be set to `DATA_AVAILABLE_STATUS`.
  - For `MethodCallProcessingMode = kPoll`, `Listener` shall remain unset, and `StatusMask` shall be set to `STATUS_MASK_NONE`.

⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_11106] Creating a DataWriter to handle method responses on the server side ⌈The DDS binding shall create a DDS DataWriter for the Reply Topic associated with the `method`s of the `ServiceInterface` (see [SWS_CM_11102]) as part of the `OfferService()` operation (see [SWS_CM_11101]).

If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, the binding implementation shall use the DDS Publisher created in [SWS_CM_11002] (whose partition name is `"ara.com://services/<svcId>_<svcInId>"`) to create the DataWriter.

The DataWriter shall be configured as follows:

- `DataWriterQos` shall be set as specified in the Manifest, where the `DdsProvidedServiceInstance` element defines the `qosProfile` that shall be used.

⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_11107] Calling a service method from the client side ⌈When instructed to call a method from the client side, the DDS binding shall construct a new sample of the Request Topic—an instance of the Request Topic data type defined in [SWS_CM_11101])—as follows:

- To initialize the `RequestHeader` object,
  - `requestId` shall be set by the underlying DDS implementation according to the rules specified in [21].
  - `instanceName` shall be set by the binding implementation to the `serviceInstanceId` of the remote service instance.
- To initialize the `<svcId>Method_Call` object, the binding implementation shall first select the appropriate union case (as specified in [SWS_CM_11101], the hash of the method's name is the union discriminator that selects the union case), and then set accordingly the structure containing all the `in` and `inout` arguments.

That sample shall then be passed as a parameter to the `write()` method of the DDS DataWriter created in [SWS_CM_11103] to handle method requests on the client side, which shall serialize the sample according to the DDS serialization rules, and publish it over DDS.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213)*

The DDS serialization rules are defined in section 7.8.3.7.

[SWS_CM_11108] Notifying the client of a response to a method call ⌈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.

If the discriminator of the `<svcId>Method_<methodName>_Result` union holding the response for the specific method call in the received DDS Reply Topic sample is `dds::RETCODE_OK` (i.e., 0 as defined in [18]), the binding implementation shall call the `ara::core::Promise`'s `set_value()` operation (see [SWS_CORE_00345] and [SWS_CORE_00346]) using the members representing the `out` and `inout` arguments in the corresponding `<svcId>Method_<methodName>_Out result` (see [SWS_CM_11102]).

Else, for any other discriminator value, the binding implementation shall call the `ara::core::Promise`'s `SetError()` operation (see [SWS_CORE_00347]) with the corresponding `ara::core::ErrorCode`, which is based on the corresponding `ApApplicationError` (see [SWS_CM_11102]).

In either case, the associated set operation shall be performed upon the reception of a new Reply Topic sample by the corresponding DDS DataReader (see [SWS_CM_11104]). The DDS binding shall use the DataReader's `take()` to process the sample. Moreover, to correlate a request with a response, the binding shall compare the `header.relatedRequestId` of the received sample with the original `requestId` that was set and sent in [SWS_CM_11107][11] [12].

If a received `relatedRequestId` does not correspond to a `requestId` that has been sent by the client, the response shall be discarded.⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215)*

[SWS_CM_11109] Processing a method call on the server side (event driven) ⌈In case a `MethodCallProcessingMode` of either `kEvent` or `kEventSingleThread` has been passed to the constructor of the `ServiceSkeleton` (see [SWS_CM_00130]), the binding implementation shall create a `DataReaderListener` to process the requests asynchronously—as described in [SWS_CM_11110]—and attach an instance of it to the DataReader processing the requests in accordance with [SWS_CM_11105]. The listener is responsible for identifying the method that shall

[11] The RPC over DDS specification [21] does not mandate a specific mechanism or context to invoke the `take()` operation on the DataReader that subscribes to method replies.Implementers of this specification may therefore follow different approaches to address this issue.

[12] For instance, a proxy could provide a `ara::core::Map<dds::SampleIdentity, ara::core::Promise<T> >` to hold the `ara::core::Promises` assigned to every request (identified by their `dds::SampleIdentity requestId`), and install a `DataReaderListener` (on the DataReader created in [SWS_CM_11104]) with an `on_data_available()` method that could call the setter of the corresponding `ara::core::Promise` using the `relatedRequestId` of the received Reply Topic sample to address it. Alternatively, a compliant solution could also call `take()` in the context of a `std::async` using a `dds::core::Waitset` [18] to block until the reception of the expected sample.

process the request and dispatch it (see [SWS_CM_11110]).⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_11110] Creating a DataReaderListener to process asynchronous requests on the server side ⌈According to [SWS_CM_11105], a `MethodCallProcessingMode` of either `kEvent` or `kEventSingleThread` requires the instantiation of a DataReaderListener to process asynchronously requests on the server side. The resulting listener shall derive from the standard `DataReaderListener` class [18], specifying that the data type of the samples to be handled is the Request Topic data type defined in [SWS_CM_11101].

The `DataReaderListener` shall implement the following methods according to the specified instructions:

- An `on_data_available()` method responsible for reading the received requests from the DataReader's cache—using the `take()` operation—and dispatching them to the appropriate methods for processing. To identify the method of the `ServiceSkeleton` class that shall process each request, `on_data_available()` shall use the union discriminator of the `<svcId>Method_Call` and provide the destination method with the specific `ArgumentDataPrototype`s in the union case.

⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_11111] Processing a method call on the server side (polling) ⌈In case a `MethodCallProcessingMode` of `kPoll` has been passed to the constructor of the `ServiceSkeleton` (see [SWS_CM_00130]), the `ProcessNextMethodCall` method is be responsible for calling `take()` on the DataReader processing the Request Topic associated with the service (see [SWS_CM_11105]). `ProcessNextMethodCall`, shall take only the first sample from the DataReader's cache and dispatch the call the appropriate service method (see [SWS_CM_00191]) of the `ServiceSkeleton` class according to the value of the of the discriminator of the `<svcId>Method_Call` union and provide the destination method with the specific `ArgumentDataPrototype`s in the union case.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_11112] Sending a method call response from the server side ⌈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]).

To send the response, the DDS binding shall construct a new sample of the Reply Topic —an instance of the Reply Topic data type defined in [SWS_CM_11102])—as follows:

- To initialize the `ReplyHeader` object,
  - `relatedRequestId` shall be set to the value of the `header.requestId` attribute of the request that triggered the method call (see [SWS_CM_11107]).

- To initialize the <svcId>Method_Return object, the binding implementation shall:
  - Select the appropriate union case (as specified in [SWS_CM_11102], the hash of the method's name is the union discriminator that selects the union case).
  - Set the <svcId>Method_<methodName>_Result union selecting its union discriminator based on whether the operation generated the correct result or raised an ApApplicationError:
    * If operation generated the correct result, the binding shall select the union case for dds::RETCODE_OK and set the <svcId>Method_<methodName>_Out structure with all the out and inout arguments.
    * Otherwise, if the operation raised an ApApplicationError, the binding shall select the union case 0x01 and construct the corresponding ara::core::ErrorCode (see [SWS_CM_11102]).

The sample shall then be passed as a parameter to the write() method of the DDS DataWriter created in [SWS_CM_11105] to handle method responses on the server side, which shall serialize the sample according to the DDS serialization rules, and publish it over DDS.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213)*

The DDS serialization rules are defined in section 7.8.3.7.

7.8.3.6 Handling Fields

[SWS_CM_11130] Mapping Fields with hasNotifier attribute to DDS Topics ⌈The DDS binding shall assign a DDS Topic to every Field defined in the ServiceInterface in the role field with hasNotifier = true to enable its notification semantics over DDS. The equivalent DDS Topic shall be configured as follows:

- The Topic Name shall be derived from the Manifest according to the following rules:
  - If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION orSERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the topic name shall be set to ara.com://services/<InterfaceID>/<Major>.<Minor>/<TopicName>
  - Additionally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this topic shall be sent and received via DataWriters and DataReaders whose respective parent Publisher and Subscriber objects include the following partition in the

PARTITION QoS policy: `ara.com://services/<InterfaceID>/<InstanceId>`

- Finally, if the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_TOPIC_PREFIX`, then the topic name shall be set to `ara.com://services/<InterfaceID>/<InstanceID>/<TopicName>`
- Where:

  <InterfaceID> is the value of `DdsServiceInterfaceDeployment.serviceInterfaceId`

  <InstanceID> is the value of either `DdsProvidedServiceInstance.serviceInstanceId` or `DdsRequiredServiceInstance.requiredServiceInstanceId`

  <Major> and <Minor> are the values of `ServiceInterface.majorVersion` and `ServiceInterface.minorVersion`, respectively

  <TopicName> is the value of `DdsEventDeployment.topicName` defined for `DdsFieldDeployment` in the `notifier` role

- The Topic Data Type shall be defined as specified in [SWS_CM_11131], and shall be registered under the equivalent data type's name.

⌋*(RS_CM_00204, RS_CM_00201)*

[SWS_CM_11131] Field Notifier DDS Topic data type definition ⌈The data type of a DDS Topic representing a Field Notifier shall be constructed according to the following IDL definition:

```
struct <fieldTypeName>FieldNotifierType {
    @key uint16 instance_id;
    <fieldTypeName> data;
};
```

Where:

<fieldTypeName> is the `Cpp Implementation Data Type symbol` [24].

instance_id is a `@key` member of the type, which identifies all samples with the same `instance_id` as samples of the same Topic Instance.

data is the actual value of the `field`, which shall be constructed and encoded according to the DDS serialization rules. The `@external` annotation is optionally allowed, for cases where references yield implementation benefits over values.

⌋*(RS_CM_00204, RS_CM_00201)*

The DDS serialization rules are defined in section 7.8.3.7.

[SWS_CM_11132] Mapping of Update method ⌈When instructed to transmit a field notification message, the DDS binding shall construct a new sample of the equivalent DDS Topic data type (see [SWS_CM_11131]) as follows:

- The Instance Id field (`instance_id`) shall be derived from the Manifest, where the `DdsProvidedServiceInstance` element defines the `serviceInstanceId`.
- The Data field (`data`) shall point to the data input parameter of the `Update()` method.

That sample shall be then passed as a parameter to the `write()` method of the DDS DataWriter associated with the `field`, which shall serialize the sample according to the DDS serialization rules specified, and publish it over DDS.⌋*(RS_CM_00204, RS_CM_00201)*

The DDS serialization rules are defined in section 7.8.3.7.

[SWS_CM_11133] Mapping of Subscribe method ⌈When instructed to subscribe to a field, the DDS binding shall create a DDS DataReader to handle the subscription using the DDS Subscriber created for the proxy in [SWS_CM_11002]. The rules to create the DataReader are specified in [SWS_CM_11134].⌋*(RS_CM_00204, RS_CM_00103)*

[SWS_CM_11134] Creating a DDS DataReader for field subscription ⌈The DDS binding shall create a DDS DataReader for the Topic associated with the `field` (see [SWS_CM_11130]). If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure the proxy communicates only with the service intsance it is bound to, the binding implementation shall use the DDS Subscriber created in [SWS_CM_11009] (whose partition name is `"ara.com://services/<svcId>_<reqSvcInId>"`) to create the DataReader.

The DataReader shall be configured as follows:

- `DataReaderQos` shall be set as specified in the Manifest, where the `DdsFieldQosProps` element defines the `qosProfile` that shall be used. To configure the DataReader's cache size according to the field subscription semantics, the `maxSampleCount` specified in the `Subscribe()` method call, the value of the DataReader's HISTORY QoS specified in `qosProfile` shall be overridden as follows:
  - `history.kind = KEEP_LAST_HISTORY_QOS`
  - `history.depth = <maxSampleCount>`

  Moreover, to ensure that the proxy received the current value of the field as soon as it creates the subscription, the DataReaders's DURABILITY QoS shall be overridden as follows:
  - `durability.kind = TRANSIENT_LOCAL_DURABILITY_QOS`

  Likewise, the RELIABILITY QoS shall be overridden as follows:

  - `reliability.kind = RELIABLE_RELIABILITY_QOS`

- `Listener` shall be an instance of the `DataReaderListener` class specified in [SWS_CM_11135].
- `StatusMask` shall be set to `STATUS_MASK_NONE`.

⌋*(RS_CM_00204, RS_CM_00103)*

[SWS_CM_11135] Creating a DDS DataReaderListener for field subscription ⌈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 following the instructions specified in [SWS_CM_11020].

The `DataReaderListener` class shall specify that the samples to be handled are of the Topic data type specified in [SWS_CM_11131].⌋*(RS_CM_00204, RS_CM_00103)*

[SWS_CM_11136] Mapping of Unsubscribe method ⌈When instructed to unsubscribe from a field event, the DDS binding shall delete the DataReader associated with the `field` notifier.⌋*(RS_CM_00204, RS_CM_00104)*

[SWS_CM_11137] Mapping of GetSubscriptionState method ⌈The GetSubscriptionState method shall be mapped as specified in [SWS_CM_11022] using the DataReader created in [SWS_CM_11134].⌋*(RS_CM_00204, RS_CM_00106)*

[SWS_CM_11138] Mapping of GetNewSamples method ⌈The GetNewSamples method shall be mapped as specified in [SWS_CM_11023] using the DataReader created in [SWS_CM_11134].⌋*(RS_CM_00204, RS_CM_00202)*

[SWS_CM_11139] Mapping of GetFreeSampleCount method ⌈The GetFreeSampleCount method shall be mapped as specified in [SWS_CM_11024] using the DataReader created in [SWS_CM_11134].⌋*(RS_CM_00204, RS_CM_00202)*

[SWS_CM_11140] Mapping of SetReceiveHandler method ⌈The SetReceiveHandler method shall be mapped as specified in [SWS_CM_11025] using the DataReader created in [SWS_CM_11134].⌋*(RS_CM_00204, RS_CM_00203)*

[SWS_CM_11141] Mapping of UnsetReceiveHandler method ⌈The UnsetReceiveHandler method shall be mapped as specified in [SWS_CM_11026] using the DataReader created in [SWS_CM_11134].⌋*(RS_CM_00204, RS_CM_00203)*

[SWS_CM_11142] Mapping of SetSubscriptionStateHandler method ⌈The SetSubscriptionStateHandler method shall be mapped as specified in [SWS_CM_11027] using the DataReader created in [SWS_CM_11134].⌋*(RS_CM_00204, RS_CM_00106)*

[SWS_CM_11143] Mapping of UnsetSubscriptionStateHandler method ⌈The UnsetSubscriptionStateHandler method shall be mapped as specified in [SWS_CM_11028] using the DataReader created in [SWS_CM_11134].⌋*(RS_CM_00204, RS_CM_00106)*

[SWS_CM_11144] Mapping of Field Get/Set methods to DDS Service Methods and Topics ⌈Every ServiceInterface containing one or more Fields defined in the role field with hasGetter or hasSetter attributes set to true shall have an associated DDS Service to enable ara::com Service Instances to offer those operations, and to enable client applications to invoke them. The equivalent DDS Service shall provide the getter and setter methods for all the fields in the corresponding ServiceInterface.

In compliance with [SWS_CM_11100], these DDS Services shall be constructed according to the Basic Service Mapping Profile of the RPC over DDS specification [21]. Thus, every ServiceInterface containing one or more fields with the hasGetter or hasSetter attributes enabled shall trigger the creation of a pair of DDS Topics: a Request Topic and a Reply Topic.

The equivalent DDS Request Topic shall be configured as follows:

- The Request Topic Name shall be derived from the Manifest according to the following rules:
  - If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION or SERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the topic name shall be set to ara.com://services/<InterfaceID>/<Major>.<Minor>/<TopicName>
  - Additionally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this topic shall be sent and received via DataWriters and DataReaders whose respective parent Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: ara.com://services/<InterfaceID>/<InstanceId>
  - Finally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_TOPIC_PREFIX, then the topic name shall be set to ara.com://services/<InterfaceID>/<InstanceID>/<TopicName>
  - Where:

    <InterfaceID> is the value of DdsServiceInterfaceDeployment.serviceInterfaceId

    <InstanceID> is the value of either DdsProvidedServiceInstance.serviceInstanceId or DdsRequiredServiceInstance.requiredServiceInstanceId

    <Major> and <Minor> are the values of ServiceInterface.majorVersion and ServiceInterface.minorVersion, respectively

<TopicName> is the value of DdsServiceInterfaceDeployment.fieldRequestTopicName

- The Request Topic Data Type shall be defined as specified in [SWS_CM_11145].

The equivalent DDS Reply Topic shall be configured as follows:

- The Reply Topic Name shall be derived from the Manifest according to the following rules:
  - If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION or SERVICE_INSTANCE_RESOURCE_INSTANCE_ID, then the topic name shall be set to ara.com://services/<InterfaceID>/<Major>.<Minor>/<TopicName>
  - Additionally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, then samples of this topic shall be sent and received via DataWriters and DataReaders whose respective parent Publisher and Subscriber objects include the following partition in the PARTITION QoS policy: ara.com://services/<InterfaceID>/<InstanceId>
  - Finally, if the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_TOPIC_PREFIX, then the topic name shall be set to ara.com://services/<InterfaceID>/<InstanceID>/<TopicName>
  - Where:

    <InterfaceID> is the value of DdsServiceInterfaceDeployment.serviceInterfaceId

    <InstanceID> is the value of either DdsProvidedServiceInstance.serviceInstanceId or DdsRequiredServiceInstance.requiredServiceInstanceId

    <Major> and <Minor> are the values of ServiceInterface.majorVersion and ServiceInterface.minorVersion, respectively

    <TopicName> is the value of DdsServiceInterfaceDeployment.fieldReplyTopicName
- The Reply Topic Data Type shall be defined as specified in [SWS_CM_11146].

⌋(*RS_CM_00204, RS_CM_00212, RS_CM_00213*)

[SWS_CM_11145] DDS Service Request Topic data type definition for Field getter and setter operations ⌈As specified in section 7.5.1.1.6 of [21], the Request Topic

data type is a structure composed of a Request Header with meta-data and a Call Structure with data. The IDL definition of the Request Topic data type for the DDS Service handling field getters and setters is the following:

```
struct <svcId>Field_Request {
    dds::rpc::RequestHeader header;
    <svcId>Field_Call data;
};
```

Where:

<svcId> is the corresponding `serviceInterfaceId`.

dds::rpc::RequestHeader is the standard Request Header defined in section 7.5.1.1.1 of [21].

<svcId>Field_Call is the union that holds the value of the input parameters of the corresponding methods, according to the rules specified in section 7.5.1.1.6 of [21].

`dds::rpc::RequestHeader` shall be constructed as specified in section 7.5.1.1.1 of [21]. On top of that, the binding implementation shall set the `instanceName` (a member of the `RequestHeader` structure that specifies the DDS service instance name) to a string representation of the `serviceInstanceId` of the service instance that provides the fields (which have getters or setters).

`<svcId>Field_Call` shall be constructed as specified in section 7.5.1.1.6 of [21].

- The name of the union shall be `<svcId>Field_Call`.
- The union discriminator shall be a 32-bit signed integer.
- The union shall have a default case of type `dds::rpc::UnknownOperation` (defined in section 7.5.1.1.1 of [21]) for unsupported and unknown operations.
- The union shall have a case label for each `hasGetter` and `hasSetter` attribute equal to `true` in the `Field`s defined in the `ServiceInterface` with the role `field`, where:
  - The integer value of the case label shall be a 32-bit hash of the field getter or setter name. That is, `"Get<fieldName>"` and `"Set<fieldName>"`; where `<fieldName>` is the `shortName` of the `Field`. The binding implementation shall compute the hash as specified in section 7.5.1.1.2 of [21]. Representations of the service interface in OMG IDL [23] shall define 32-bit signed integer constants (i.e., `const int32 <svcId>Field_Get<fieldName>_Hash` or `const int32 <svcId>Field_Set<fieldName>_Hash`) to simplify the representation of the union cases (see below).
  - The member name for the case label shall be `get<FieldName>` for getter methods and `set<FieldName>` for setter methods.

- The type for each case level shall be `<svcId>Field_Get<fieldName>_In` for getter methods, and `<svcId>Field_Set<fieldName>_In` for setter methods, which shall be constructed as specified in section 7.5.1.1.4 of [21] (see below).

The IDL definition of the `<svcId>Field_Call` union is the following:

```
union <svcId>Field_Call switch(int32) {
default:
    dds::rpc::UnknownOperation unknownOp;
case <svcId>Field_Get<Field0Name>_Hash:
    <svcId>Field_Get<Field0Name>_In get<Field0Name>;
case <svcId>Field_Set<Field0Name>_Hash:
    <svcId>Field_Set<Field0Name>_In set<Field0Name>;
case <svcId>Field_Get<Field1Name>_Hash:
    <svcId>Field_Get<Field1Name>_In get<Field1Name>;
case <svcId>Field_Set<Field1Name>_Hash:
    <svcId>Field_Set<Field1Name>_In set<Field1Name>;
// ...
case <svcId>Field_Get<FieldNName>_Hash:
    <svcId>Field_Get<FieldNName>_In get<FieldNName>;
case <svcId>Field_Set<FieldNName>_Hash:
    <svcId>Field_Set<FieldNName>_In set<FieldNName>;
};
```

According to 7.5.1.1.4 of [21], `<svcId>Field_Set<FieldName>_In` structures shall contain as member, the corresponding `StdCppImplementationDataType` representing the value of `Field` to be set. Conversely, `<svcId>Field_Get<FieldName>_In` shall contain a single member named `dummy` of type `dds::rpc::UnusedMember` (see section 7.5.1.1.1 of [21]) to indicate that the method has no input parameters.

The resulting Request Topic data type shall be encoded according to the DDS serialization rules. Unions, such as the `<svcId>Field_Call` union, shall be serialized as specified in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_11146] DDS Service Reply Topic data type definition for Field getter and setter operations ⌈As specified in section 7.5.1.1.7 of [21], the Reply Topic data type is a structure composed of a Reply Header with meta-data and a Return Structure with data. The IDL definition of the Reply Topic data type for the DDS Service handling field getters and setters is the following:

```
struct <svcId>Field_Reply {
    dds::rpc::ReplyHeader header;
    <svcId>Field_Return data;
};
```

Where:

`<svcId>` is the corresponding `serviceInterfaceId`.

`dds::rpc::ReplyHeader` is the standard Reply Header defined in section 7.5.1.1.1 of [21].

<svcId>Field_Return is the union that holds the return values of the corresponding response, according to the rules specified in section 7.5.1.1.7 of [21].

`dds::rpc::ReplyHeader` shall be constructed as specified in section 7.5.1.1.1 of [21].

`<svcId>Field_Return` shall be constructed as specified in section 7.5.1.1.7 of [21]:

- The name of the union shall be `<svcId>Field_Return`.
- The union discriminator shall be a 32-bit signed integer.
- The union shall have a default case of type `dds::rpc::UnknownOperation` (defined in section 7.5.1.1.1 of [21]) for unsupported and unknown operations.
- The union shall have a case label for each `hasGetter` and `hasSetter` attribute equal to `true` in the `Field`s defined in the `ServiceInterface` with the role `field`, where:
  - The integer value of the case label shall be a 32-bit hash of the field getter or setter name. That is, `"Get<FieldName>"` and `"Set<FieldName>"`; where `<FieldName>` is the `shortName` of the `Field`. The binding implementation shall compute the hash as specified in section 7.5.1.1.2 of [21]. Representations of the service interface in OMG IDL [23] shall define 32-bit signed integer constants (i.e., `const int32 <svcId>Field_Get<FieldName>_Hash` or `const int32 <svcId>Field_Set<FieldName>_Hash`) to simplify the representation of the union cases (see below).
  - The member name of the case label shall be `get<FieldName>` for getter methods and `set<FieldName>` for setter methods.
  - The type for each case label shall be `<svcId>Field_Get<FieldName>_Result` for getter methods and `<svcId>Field_Set<FieldName>_Result` for setter methods, which shall be constructed as specified in section 7.5.1.1.4 of [21] (see below).

The IDL definition of `<svcId>Field_Return` is the following:

```
union <svcId>Field_Return switch(int32) {
default:
    dds::rpc::UnknownOperation unknownOp;
case <svcId>Field_Get<Field0Name>_Hash:
    <svcId>Field_Get<Field0Name>_Result get<Field0Name>;
case <svcId>Field_Set<Field0Name>_Hash:
    <svcId>Field_Set<Field0Name>_Result set<Field0Name>;
case <svcId>Field_Get<Field1Name>_Hash:
    <svcId>Field_Get<Field1Name>_Result get<Field1Name>;
case <svcId>Field_Set<Field1Name>_Hash:
    <svcId>Field_Set<Field1Name>_Result set<Field1Name>;
// ...
case <svcId>Field_Get<FieldNName>_Hash:
    <svcId>Field_Get<FieldNName>_Result get<FieldNName>;
case <svcId>Field_Set<FieldNName>_Hash:
```

```
    <svcId>Field_Set<FieldNName>_Result set<FieldNName>;
};
```

According with [SWS_CM_00112] and [SWS_CM_00113], both getters and setters have the same output parameter. Therefore, in accordance with section 7.5.1.1.5 of [21], both the `<svcId>Field_Get<FieldName>_Result` and `<svcId>Field_Set<FieldName>_Result` unions shall be constructed as follows:

- The union discriminator shall be a 32-bit signed integer.
- The union shall have a case with label `dds::RETCODE_OK` to represent a successful return:
  - The value of `RETCODE_OK` shall be 0, as specified in section 2.3.3 of [18].
  - The successful case shall have a single member named `result_` of type `<svcId>Field_Get<FieldName>_Out` to hold the value to be returned to the getter, or type `<svcId>Field_Set<FieldName>_Out` to hold the value to be returned to the setter (see below).

The IDL representation of `<svcId>Field_Get<FieldName>_Result` is the following:

```
union <svcId>Field_Get<FieldName>_Result switch(int32) {
case dds::RETCODE_OK:
    <svcId>Field_Get<FieldName>_Out result_;
};
```

Likewise, the IDL representation of `<svcId>Field_Set<FieldName>_Result` is the following:

```
union <svcId>Field_Set<FieldName>_Result switch(int32) {
case dds::RETCODE_OK:
    <svcId>Field_Set<FieldName>_Out result_;
};
```

Both types `<svcId>Field_Get<FieldName>_Out` and its counterpart `<svcId>Field_Set<FieldName>_Out` shall map to a structure with a single member named `return_` of the `StdCppImplementationDataType` representing the value of the corresponding `Field`.

The resulting Reply Topic data type shall be encoded according to the DDS serialization rules. Unions, such as the `<svcId>Field_Return` union, shall be serialized as specified in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213)*

[SWS_CM_11147] Creating a DataWriter to handle get/set requests on the client side ⌈The DDS binding shall create a DDS DataWriter for the Request Topic associated with the getters and setters of the `field`s of the `ServiceInterface` (see [SWS_CM_11145]) upon proxy instantiation.

If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, to ensure

the proxy communicates only with the service instance it is bound to, the binding implementation shall use the DDS Publisher created in [SWS_CM_11009] (whose partition name is "ara.com://services/<svcId>_<reqSvcInId>") to create the DataWriter.

The DataWriter shall be configured as follows:

- DataWriterQos shall be set as specified in the Manifest, where the DdsFieldQosProps element defines the qosProfile that shall be used.

⌋(*RS_CM_00204, RS_CM_00212, RS_CM_00213*)

[SWS_CM_11148] Creating a DataReader to handle get/set responses on the client side ⌈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]) upon proxy instantiation.

If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, to ensure the proxy communicates only with the service instance it is bound to, the binding implementation shall use the DDS Subscriber created in [SWS_CM_11009] (whose partition name is "ara.com://services/<svcId>_<reqSvcInId>") to create the DataReader.

The DataReader shall be configured as follows:

- DataReaderQos shall be set as specified in the Manifest, where the DdsFieldQosProps element defines the qosProfile that shall be used.

⌋(*RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00215*)

[SWS_CM_11149] Creating a DataReader to handle get/set requests on the server side ⌈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_11145]).

If the provided or consumed Service Instance has been advertised with the identifier_type attribute set to SERVICE_INSTANCE_RESOURCE_PARTITION, the binding shall use the DDS Subscriber created in [SWS_CM_11002] (whose partition name is "ara.com://services/<svcId>_<svcInId>") to create the DataReader.

The DataReader shall be configured as follows:

- DataReaderQos shall be set as specified in the Manifest, where the DdsFieldQosProps element defines the qosProfile that shall be used.
- Listener and StatusMask shall be set according to the value of MethodCallProcessingMode that was selected in the constructor of the ServiceSkeleton class:

- For `MethodCallProcessingMode = kEvent` or `kEventSingleThread`, `Listener` shall be set to an instance of the `DataReaderListener` class specified in [SWS_CM_11154], and `StatusMask` shall be set to `DATA_AVAILABLE_STATUS`.
- For `MethodCallProcessingMode = kPoll`, `Listener` shall remain unset, and `StatusMask` shall be set to `STATUS_MASK_NONE`.

⌋(*RS_CM_00204, RS_CM_00212, RS_CM_00213*)

[SWS_CM_11150] Creating a DataWriter to handle get/set responses on the server side ⌈The DDS binding shall create a DDS DataWriter for the Reply Topic associated with the getters and setters of the `field`s of the `ServiceInterface` (see [SWS_CM_11146]).

If the provided or consumed Service Instance has been advertised with the `identifier_type` attribute set to `SERVICE_INSTANCE_RESOURCE_PARTITION`, the binding implementation shall use the DDS Publisher created in [SWS_CM_11002] (whose partition name is `"ara.com://services/<svcId>_<svcInId>"`) to create the DataWriter.

The DataWriter shall be configured as follows:

- `DataWriterQos` shall be set as specified in the Manifest, where the `DdsFieldQosProps` element defines the `qosProfile` that shall be used.

⌋(*RS_CM_00204, RS_CM_00212, RS_CM_00213*)

[SWS_CM_11151] Calling get/set method associated with a field from the client side ⌈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—an instance of the Request Topic data type defined in [SWS_CM_11145]—as follows:

- To initialize the `RequestHeader` object,
  - `requestId` shall be set by the underlying DDS implementation according to the rules specified in [21].
  - `instanceName` shall be set by the binding implementation to the `serviceInstanceId` of the remote service instance.
- To initialize the `<svcId>Field_Call` object, the binding implementation shall first select the appropriate union case (as specified in [SWS_CM_11145], the hash of the field getter/setter's name is the union discriminator that selects the union case). Then,
  - If the call corresponds to a getter, the binding shall leave the `dummy` member of the `<svcId>Field_Get<FieldName>_In` structure unset.

– Else, if the call corresponds to a setter, the binding shall set accordingly the only member of the <svcId>Field_Set<FieldName>_In structure with the new value for the field.

That sample shall then be passed as a parameter to the write() method of the DDS DataWriter created in [SWS_CM_11147] to handle get/set requests on the client side, which shall serialize the sample according to the DDS serialization rules, and publish it over DDS.⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218)*

The DDS serialization rules are defined in section 7.8.3.7.

[SWS_CM_11152] Notifying the client of the response to the get/set method call ⌈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 value of the corresponding result_ member of either the <svcId>Field_Get<FieldName>_Result structure, for get operations; or <svcId>Field_Set<FieldName>_Out, for set operations.

The associated set operation shall be performed upon the reception of a new Reply Topic sample by the corresponding DDS DataReader (see [SWS_CM_11148]). The DDS binding shall use the DataReader's take() method to process the sample. Moreover, to correlate a request with a response, the binding shall compare the header.relatedRequestsId of the received sample with the original requestId that was sent in [SWS_CM_11151][13]. If the relatedRequestId does not correspond to a requestId that has been sent by the client, the response shall be discarded.⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00217, RS_CM_00218)*

[SWS_CM_11153] Processing a get/set method call associated with a field on the server side (event driven) ⌈In case a MethodCallProcessingMode of either kEvent or kEventSingleThread has been passed to the constructor of the ServiceSkeleton (see [SWS_CM_00130]), the binding implementation shall create a DataReaderListener to process the requests asynchronously—as described in [SWS_CM_11154]—and attach an instance of it to the DataReader processing the requests for the getters and setters of the ServiceInterface's fields in accordance with [SWS_CM_11149]. The listener is responsible for identifying the method that shall process the request and dispatch it (see [SWS_CM_11154]).⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221)*

[SWS_CM_11154] Creating a DataReaderListener to process asynchronous requests for field getters and setters on the server side ⌈According to [SWS_CM_11149], a MethodCallProcessingMode of either kEvent or kEventSingleThread requires the instantiation of a DataReaderListener to process asynchronously requests on the server side. The resulting listener shall derive from the standard DataReaderListener class [18], specifying that the type of the samples to be handled is the Request Topic data type defined in [SWS_CM_11145].

[13] See footnotes in [SWS_CM_11108].

The DataReaderListener shall implement the following method according to the specified instructions:

- An on_data_available() method responsible for reading the received requests from the DataReader's cache—using the take() operation—and dispatching it to the corresponding registered SetHandler or—if it applies—GetHandler (see [SWS_CM_00114] and [SWS_CM_00116]). To identify the field of the ServiceSkeleton class, the operation (i.e., Set() or Get()), and therefore the corresponding handler; on_data_available() shall use the union discriminator of the <svcId>Field_Call union (see [SWS_CM_11145]). In the case of a Set() operation, the method shall provide the corresponding SetHandler with the only member of the received <svcId>Field_<FieldName>_In structure, which contains the new value to be set. In the case of a Get() operation, the binding shall dispatch to the corresponding GetHandler—if it was registered—or to an internal lookup operation for the current value of the field if it was not.

⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221)*

[SWS_CM_11155] Processing a get/set method call associated with a field on the server side (polling) ⌈In case a MethodCallProcessingMode of kPoll has been passed to the constructor of the ServiceSkeleton (see [SWS_CM_00130]), the ProcessNextMethodCall method is responsible for calling take() on the DataReader processing the Request Topic associated with the service (see [SWS_CM_11145]). ProcessNextMethodCall shall take only the first sample from the DataReader's cache and dispatch it to the corresponding registered SetHandler or—if it applies—GetHandler (see [SWS_CM_00114] and [SWS_CM_00116]).

To identify the field of the ServiceSkeleton class, the operation (i.e., Set() or Get()), and therefore the corresponding handler, the binding implementation shall use the union discriminator of the <svcId>Field_Call union (see [SWS_CM_11145]). In the case of a Set() operation, the binding shall provide the corresponding SetHandler with the only member of the received <svcId>Field_<FieldName>_In structure, which contains the new value to be set. In the case of a Get() operation, the binding shall call the corresponding GetHandler—if it was registered—or dispatch to an internal lookup operation for the current value of the field if it was not.⌋ *(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221)*

[SWS_CM_11156] Sending a response for a get/set method call associated with a field from the server side ⌈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[14] as a result of a Get() operation where no GetHandler was previously registered.

[14] An internal lookup operation to retrieve the current value of a field.

To send the response, the DDS binding shall construct a new sample of the Reply Topic—an instance of the Reply Topic data type defined in [SWS_CM_11146]—as follows:

- To initialize the `ReplyHeader` object,
  - `relatedRequestId` shall be set to the value of the `header.requestId` attribute of the request that triggered the method call (see [SWS_CM_11151]).
- To initialize the `<svcId>Field_Return` object, the binding implementation shall:
  - Select the appropriate union case (as specified in [SWS_CM_11146]), the hash of the field's getter/setter method is the union discriminator that selects the union case).
  - Set the appropriate `<svcId>Field_Get<FieldName>_Result`—for `Get()` operations—or `<svcId>Field_Set<FieldName>_Result`—for `Set()` operations. In both cases, the binding shall select the union case for `dds::RETCODE_OK` and set the corresponding structure with the value retrieved upon the return of (1), (2), or (3).

The sample shall then be passed as a parameter to the `write()` method of the DDS DataWriter created in [SWS_CM_11150] to handle method responses on the server side, which shall serialize the sample according to the DDS serialization rules, an publish it over DDS.⌋*(RS_CM_00204, RS_CM_00212, RS_CM_00213, RS_CM_00220, RS_CM_00221)*

The DDS serialization rules are defined in section 7.8.3.7.

7.8.3.7 Serialization of Payload

[SWS_CM_11040] DDS standard serialization rules ⌈The serialization of the payload shall be done according to the DDS standard serialization rules defined in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00201)*

7.8.3.7.1 Basic Data Types

[SWS_CM_11041] DDS serialization of `StdCppImplementationDataType` of `category` VALUE ⌈`StdCppImplementationDataType` of `category` VALUE shall be serialized according to the standard serialization rules for the equivalent DDS `PRIMITIVE_TYPE` defined in section 7.4.3.5 of [20]. Table 7.4 provides the equivalent DDS `PRIMITIVE_TYPE`s for the primitive `StdCppImplementationDataType`s with `category` VALUE defined in [13].⌋*(RS_CM_00204, RS_CM_00200, RS_CM_00102)*

Type	DDS Type	Remark

boolean	Boolean	
std::uint8_t	Byte	Shall be encoded as a Byte type (opaque 8-bit type).
std::uint16_t	UInt16	
std::uint32_t	UInt32	
std::uint64_t	UInt64	
std::int8_t	Byte	Shall be encoded as a Byte type (opaque 8-bit type).
std::int16_t	Int16	
std::int32_t	Int32	
std::int64_t	Int64	
float	Float32	
double	Float64	

Table 7.4: `StdCppImplementationDataType`s with category VALUE supported for serialization

7.8.3.7.2 Enumeration Data Types

[SWS_CM_11042] DDS serialization of enumeration data types ⌈Enumeration data types shall be serialized according to the standard serialization rules for DDS `ENUM_TYPE` defined in section 7.4.3.5 of [20].

The bit bound of the `ENUM_TYPE` shall be set to the size of the enumeration's underlying basic data type (i.e., the `Primitive Cpp Implementation Data Type` according to [SWS_LBAP_00027]) in bits.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.8.3.7.3 Structured Data Types (structs)

[SWS_CM_11043] DDS serialization of `StdCppImplementationDataType` of `category` `STRUCTURE` ⌈`StdCppImplementationDataType` of `category` `STRUCTURE` shall be serialized according to the standard serialization rules for DDS `STRUCT_TYPE` defined in section 7.4.3.5 of [20].

Optional members of the structure shall be marked as optional as specified in section 7.2.2.4.4.5 of [20].⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.8.3.7.4 Strings

[SWS_CM_11044] DDS serialization of `StdCppImplementationDataType` of `category` `STRING` with string `shortName` ⌈An `StdCppImplementationDataType` of `category` `STRING` shall be serialized according to the standard serialization rules for DDS `STRING_TYPE` defined in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_11046] Encoding Format and Endianness of Strings in DDS ⌈Section 7.4.1.1.2 of [20] specifies the standard character encoding format for STRING_TYPE: UTF-8. The serialized version shall not include a Byte Order Mark (BOM), as byte order information is already available in the RTPS Encapsulation Identifier and the XCDR serialization format [20].⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211, RS_AP_00136)*

7.8.3.7.5 Vectors and Arrays

[SWS_CM_11047] DDS serialization of CppImplementationDataType of category VECTOR ⌈A CppImplementationDataType of category VECTOR shall be serialized according to the standard serialization rules for DDS SEQUENCE_TYPE defined in section 7.4.3.5 of [20].

Binding implementations shall serialize VECTOR CppImplementationDataTypes with more than one dimension, as nested DDS sequences.⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

[SWS_CM_11048] DDS serialization of CppImplementationDataType of category ARRAY ⌈A CppImplementationDataType of category ARRAY shall be serialized according to the standard serialization rules for DDS ARRAY_TYPE defined in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.8.3.7.6 Associative Maps

[SWS_CM_11049] DDS serialization of CppImplementationDataType of category ASSOCIATIVE_MAP ⌈CppImplementationDataType of category ASSOCIATIVE_MAP shall be serialized according to the standard serialization rules for DDS MAP_TYPE defined in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.8.3.7.7 Variant

[SWS_CM_11050] DDS serialization of CppImplementationDataType of category VARIANT ⌈CppImplementationDataType of category VARIANT shall be serialized according to the standard serialization rules for DDS UNION_TYPE defined in section 7.4.3.5 of [20].⌋*(RS_CM_00204, RS_CM_00201, RS_CM_00202, RS_CM_00211)*

7.8.3.8 End-to-end communication protection

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 7.2 and 7.3 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 [4] (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: submessage 64-bit sequence number, as defined in [19] section 8.3.5.4 "SequenceNumber", and additional SequenceNumber-typed fields in section 8.3.7 "RTPS Submessages"
- Delay of information and blocking access to a communication channel: `LATENCY_BUDGET` Quality of Service policy, as defined in [18] section 2.2.3.8 "LATENCY_BUDGET"
- Masquerade or incorrect addressing of information: DDS Security authentication plugin, as defined in [25] section 8.3 "Authentication Plugin"
- Corruption of information, asymmetric information sent from a sender to multiple receivers: `rtpsMessageChecksum` under HeaderExtension submessage ([RTPS 2.5 or higher]). In absence of this feature, [25] also provides message integrity verification built into its message authentication protocol
- 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)

7.9 Security

In the following chapter the behavior according to the meta-model of access control and secure communication shall be described.

7.9.1 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 7.34 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.

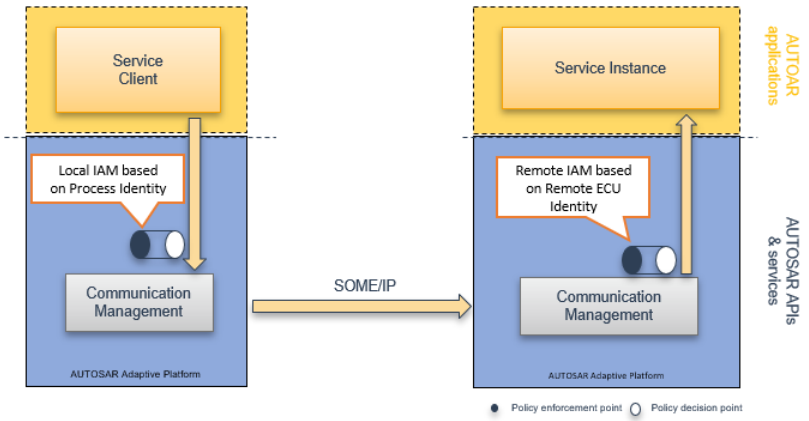


Figure 7.34: Local and Remote Identity and Access Management

The following assumptions have 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.
2. Process separation as defined in SWS IdentityAndAccessManagement [26].

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.

7.9.1.1 Configuration of Access Control

While Identity and Access Management (IAM) serves as an umbrella for access control on the Adaptive Platform, the enforcement of access control is implemented in different functional clusters such as CM. If no IAM Functional Cluster is instantiated on a Machine, then no enforcement of access control by CM is expected.

[SWS_CM_10492]{DRAFT} **IAM Module Instantiation** ⌈If no IamModuleInstantiation is defined on the Machine, CM shall perform no access control, i.e., no access to any service shall be restricted because of missing ComGrants.⌋*(RS_IAM_00002)*

Depending on the architecture and the security model, all local Processes 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]{DRAFT} **Local Access Control Activation** ⌈If IamModuleInstantiation.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 ComGrants. If IamModuleInstantiation is defined on the Machine and IamModuleInstantiation.localComAccessControlEnabled is not defined or is set to true, CM shall perform local access control.⌋*(RS_IAM_00002)*

[SWS_CM_10494]{DRAFT} **Remote Access Control Activation** ⌈If IamModuleInstantiation.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 ComGrants. If IamModuleInstantiation is defined on the Machine and IamModuleInstantiation.remoteAccessControlEnabled is not defined or is set to true, CM shall perform remote access control.⌋*(RS_IAM_00002)*

[SWS_CM_90001]{DRAFT} **Restrictions on executing methods** ⌈The invocation of a method by an application shall be executed depending on the existence of ComMethodGrant, ComFieldGrant without a reference remoteSubject and with the role attribute of ComFieldGrant set to FieldAccessEnum.getter or FieldAccessEnum.setter. From a temporal perspective the enforcement of intent shall take place between the invocation of one of the following methods and invocation of the continuation registered with then() (see [SWS_CORE_00331]) or the access to result of the Future (via the get() method (see [SWS_CORE_00326])) returned by these methods:

- the function call operator (`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 software tries to access a field/event/method in the absence of a `Grant` that controls access to the field/even/method then the error code `ComErrc::kGrantEnforcementError` shall be returned in the `Future` of the respective methods (operator(), Set(), Get()). The error shall be logged.⌋*(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

[SWS_CM_90002]{DRAFT} **Restrictions on sending events** ⌈Sending an event by an application shall be enabled depending on the existence of `ComEventGrant` or `ComFieldGrant` without a reference `remoteSubject` and with the role attribute set to `FieldAccessEnum.setter`. From a temporal perspective the enforcement of intent shall take place after the invocation of the following method:

- the `Send()` method of the respective `Event` class (see [SWS_CM_00162])
- the `Update()` method of the respective `Field` class (see [SWS_CM_00119])

A failure of the `Grant` enforcement (i.e., the triggering of an event without appropriate intent modeling) shall cause the event to be dropped silently.⌋*(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

[SWS_CM_90003]{DRAFT} **Restrictions on receiving events** ⌈Subscribing to event notifications shall be enabled depending on the existence of `ComEventGrant` or `ComFieldGrant` without a reference `remoteSubject` and with the role attribute set to `FieldAccessEnum.getter`. From a temporal perspective the enforcement of the intent shall take place after the invocation of the following method:

- the `Subscribe()` method of the respective `Event` class (see [SWS_CM_00141])

A failure of the `Grant` enforcement (i.e., the subscription to an event without appropriate intent modeling) shall cause the subscription to the event to be dropped silently.⌋*(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

[SWS_CM_10538]{DRAFT} **Restrictions on sending triggers** ⌈Sending a trigger by an application shall be enabled depending on the existence of `ComEventGrant` without a reference `remoteSubject`. In case of a `Trigger` the `ComEventGrant` references the `ServiceEventDeployment` that in turn references the `trigger`. From a temporal perspective the enforcement of intent shall take place after the invocation of the following method:

- the `Send()` method of the respective `Trigger` class (see [SWS_CM_00721])

A failure of the `Grant` enforcement (i.e., the triggering of a trigger without appropriate intent modeling) shall cause the trigger to be dropped silently.⌋*(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

[SWS_CM_10539]{DRAFT} **Restrictions on receiving triggers** ⌈Receiving a trigger shall be enabled depending on the existence of `ComEventGrant` without a reference `remoteSubject`. In case of a `Trigger` the `ComEventGrant` references the `ServiceEventDeployment` that in turn references the `trigger`. From a temporal perspective the enforcement of the intent shall take place after the invocation of the following method:

- the `Subscribe()` method of the respective `Trigger` class (see [SWS_CM_00723])

A failure of the `Grant` enforcement (i.e., the subscription to a trigger without appropriate intent modeling) shall cause the subscription to the trigger to be dropped silently.⌋ *(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

[SWS_CM_90005]{DRAFT} **Restrictions on offering services** ⌈Offering a service instance shall be enabled depending on the presence of a `ComOfferServiceGrant` without a reference `remoteSubject`. From a temporal perspective the enforcement of the intent shall take place after the invocation of the following method:

- the constructor of the respective `ServiceSkeleton` class (see [SWS_CM_00130])

If the software tries to access a field/event/method in the absence of a `Grant` that controls access to the field/even/method then the error code `ComErrc::kGrantEnforcementError` shall be returned in the Result of the named constructor function Create() for the `ServiceSkeleton` class. The error shall be logged.⌋ *(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

[SWS_CM_90006]{DRAFT} **Restrictions on using services** ⌈Using a service instance shall be enabled depending on the presence of a `ComFindServiceGrant` without a reference `remoteSubject`. From a temporal perspective the enforcement of the intent shall take place after the invocation of the following method:

- the constructor of the respective `ServiceProxy` class (see [SWS_CM_00131])

If the software tries to access a field/event/method in the absence of a `Grant` that controls access to the field/even/method then the error code `ComErrc::kGrantEnforcementError` shall be returned in the Result of the named constructor function Create() for the `ServiceProxy` class. The error shall be logged.⌋ *(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

[SWS_CM_90007] Restrictions on using RawDataStreams ⌈Using a RawDataStream instance shall be enabled depending on the presence of a `RawDataStreamGrant`. From a temporal perspective the enforcement of the intent shall take place after the invocation of the following method:

- the `Connect()` method of the respective `RawDataStream` class (see [SWS_CM_10484])

If the software tries to access a field/event/method in the absence of a `Grant` that controls access to the field/even/method then the error code `ComErrc::kGrantEnforcementError` shall be returned in the Result of the Connect() function. The error shall be logged.⌋*(RS_IAM_00006, RS_IAM_00007, RS_IAM_00010)*

Note:
In case of [SWS_CM_90002] and [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_90002], [SWS_CM_90003], [SWS_CM_90005] and [SWS_CM_90006] is up to the implementation or specific ECU projects.

7.9.1.2 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]{DRAFT} **TLS-based Authentication** ⌈Communication Management shall associate remote subjects communicating via an established (D)TLS connection to a `TlsIamRemoteSubject` according to [TPS_MANI_TlsRemotePeer].⌋ *(RS_SEC_04003)*

[SWS_CM_10496]{DRAFT} **IP and IPsec-based Authentication** ⌈Communication Management shall associate remote subjects communicating via IP to an `IPSecIamRemoteSubject` or an `IpIamRemoteSubject` according to [TPS_MANI_IPSecRemotePeer] and [TPS_MANI_IPGeneralRemotePeer].⌋*(RS_SEC_04003)*

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]{DRAFT} **Authentication Failure** ⌈If `IamModuleInstantiation.remoteAccessControlEnabled` is set to true and a remote subject cannot be authenticated, Communication Management shall silently drop all messages from this remote subject.⌋*(RS_SEC_04003)*

[SWS_CM_10498]{DRAFT} **Remote access control on executing methods** ⌈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.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10499]{DRAFT} **Remote access control on providing methods** ⌈If the execution of a method of a service interface provided by a remote subject is requested, but there exists no `ComMethodGrant` that

- references the providing remote subject in the role `remoteSubject` and
- references a `RequiredApServiceInstance` in the role `serviceInstance` and
- references the requested method in the role `serviceDeployment`,

then Communication Management shall drop the request.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10500]{DRAFT} **Remote access control on providing events** ⌈If a remote subject provides an event of a service interface, but there exists no `ComEventGrant` that

- references the providing remote subject in the role `remoteSubject` and
- references a `RequiredApServiceInstance` in the role `serviceInstance` and
- references the provided event in the role `serviceDeployment`,

then Communication Management shall drop the provided event.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10501]{DRAFT} **Remote access control on consuming events** ⌈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.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10502]{DRAFT} **Remote access control on providing field notifiers** ⌈If a remote subject sends a field notifier, but there exists no `ComFieldGrant` that

- references the providing remote subject in the role `remoteSubject` and
- references a `RequiredApServiceInstance` in the role `serviceInstance` and
- references the event in the role `serviceDeployment`,

then Communication Management shall drop the field notifier.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10503]{DRAFT} **Remote access control on providing field setters** ⌈If the execution of a set method of a field provided by a remote subject is requested, but there exists no `ComFieldGrant` that

- has the parameter `ComFieldGrant.role` set to `setter` or `getterSetter` and
- references the providing remote subject in the role `remoteSubject` and
- references a `RequiredApServiceInstance` in the role `serviceInstance` and
- references the event in the role `serviceDeployment`,

then Communication Management shall drop the request.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10504]{DRAFT} **Remote access control on providing field getters** ⌈If the execution of a get method of a field provided by a remote subject is requested, but there exists no `ComFieldGrant` that

- has the parameter `ComFieldGrant.role` set to `getter` or `getterSetter` and
- references the providing remote subject in the role `remoteSubject` and
- references a `RequiredApServiceInstance` in the role `serviceInstance` and
- references the event in the role `serviceDeployment`,

then Communication Management shall drop the request.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10505]{DRAFT} **Remote access control on consuming field notifiers** ⌈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.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10506]{DRAFT} **Remote access control on calling field setters** ⌈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.⌋*(RS_IAM_00001, RS_IAM_00002)*

[SWS_CM_10507]{DRAFT} **Remote access control on calling field getters** ⌈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.⌋*(RS_IAM_00001, RS_IAM_00002)* **[SWS_CM_10540]**{DRAFT} **Remote access control on providing triggers** ⌈If a remote subject provides a trigger of a service interface, but there exists no ComEventGrant that

- references the providing remote subject in the role remoteSubject and
- references a RequiredApServiceInstance in the role serviceInstance and
- references the ServiceEventDeployment in the role serviceDeployment that in turn references the provided trigger.

then Communication Management shall drop the provided trigger.⌋(*RS_IAM_00001, RS_IAM_00002*)

[SWS_CM_10541]{DRAFT} **Remote access control on consuming triggers** ⌈If a remote subject subscribes to an `trigger` 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 `ServiceEventDeployment` in the role `serviceDeployment` that in turn references the subscribed `trigger`.

then Communication Management shall drop the subscription request.⌋(*RS_IAM_00001, RS_IAM_00002*)

7.9.2 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:

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

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 [25] (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.

7.9.2.1 Creation and use of secure channels

7.9.2.1.1 SOME/IP and DDS network binding

[SWS_CM_90101]{DRAFT} **Secure UDP and TCP channel creation for TLS, DTLS and SecOC** ⌈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.⌋*(RS_SEC_04001)*

[SWS_CM_90102]{DRAFT} **Using secure TLS, DTLS and SecOC channels** ⌈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`.⌋*(RS_SEC_04001, RS_SEC_04003)*

The actual secure channel to be created is determined by the concrete sub-class of the `SecureComProps` base-class.

[SWS_CM_90201]{DRAFT} **Secure TLS and DTLS channel creation in the DDS Network Binding** ⌈Secure channels shall be created as specified in [SWS_CM_90101].⌋*(RS_SEC_04001)*

[SWS_CM_90202]{DRAFT} **Using TLS and DTLS secure channels in the DDS Network Binding** ⌈Secure channels shall be used as specified in [SWS_CM_90102].⌋ *(RS_SEC_04001, RS_SEC_04003)*

7.9.2.1.2 Raw data streaming

[SWS_CM_90211] Secure UDP and TCP channel creation for TLS and DTLS ⌈The Communication Management software shall create secure UDP and TCP channels according to the input for all `TlsSecureComProps` as part of the `EthernetRawDataStreamMapping`.⌋*(RS_SEC_04001)*

[SWS_CM_90212] Using secure TLS, DTLS channels ⌈All communication triggered by a `RawDataStream` shall be sent via the respective secure channel according to the input. The appropriate secure channel is defined in the `TlsSecureComProps` as part of the `EthernetRawDataStreamMapping` that is mapped to an `EthernetCommunicationConnector`.⌋*(RS_SEC_04001, RS_SEC_04003)*

7.9.2.2 (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 as well as with raw data streaming.

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

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

7.9.2.2.1 SOME/IP Network binding

[SWS_CM_90103]{DRAFT} **TLS secure channel for ServiceInterface content using reliable transport** ⌈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.⌋*(RS_SEC_04001)*

[SWS_CM_90104]{DRAFT} **DTLS secure channel for ServiceInterface content using unreliable transport** ⌈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.⌋*(RS_SEC_04001)*

[SWS_CM_90121]{DRAFT} **TLS server role of a Skeleton** ⌈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`.⌋*(RS_SEC_04001)*

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]{DRAFT} **Behavior of a creating ServiceProxy over TLS or DTLS** ⌈The instantiation according to [SWS_CM_00131] shall trigger the asynchronous handshake.⌋*(RS_SEC_04004)*

[SWS_CM_90111]{DRAFT} **Behavior of a ServiceProxy over TLS before successful completion of the handshake** ⌈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 function call operator (`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 `Future` of the respective methods (operator(), Set(), Get()). The error shall be logged.⌋*(RS_SEC_04004)*

[SWS_CM_90112]{DRAFT} **Behavior of a ServiceProxy over DTLS before successful completion of the handshake** ⌈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.⌋*(RS_SEC_04004)*

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]{DRAFT} **Behavior of a ServiceSkeleton over TLS before successful completion of the handshake** ⌈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])

⌋*(RS_SEC_04004)*

[SWS_CM_90114]{DRAFT} **Behavior of a ServiceSkeleton over DTLS before successful completion of the handshake** ⌈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])

⌋*(RS_SEC_04004)*

7.9.2.2.2 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 [19] 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, [19] defines

a UDP PSM, and different DDS vendors have implemented TCP PSMs[15], and Shared Memory PSMs for Inter-Process Communication (IPC).

For consistency with the secure channel modeling and secure communication mechanisms specified in 7.9.2.2.1, 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]{DRAFT} **TLS secure channel for methods using reliable transport** ⌈A TLS secure channel shall be created and used if:

- a `TlsSecureComProps` instance is referenced in the role `secureComPropsForTcp` 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.⌋*(RS_SEC_04001)*

[SWS_CM_90204]{DRAFT} **DTLS secure channel for methods using unreliable transport** ⌈A DTLS secure channel shall be created and used if:

- a `TlsSecureComProps` 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` 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.⌋*(RS_SEC_04001)*

[SWS_CM_90205]{DRAFT} **TLS secure channel for events using reliable transport** ⌈A TLS secure channel shall be created and used if:

- A `TlsSecureComProps` instance is referenced in the role `secureComPropsForTcp` 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`.

[15] 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`.

The DataReaders and DataWriters associated with the event shall be configured to operate over TLS.⌋(*RS_SEC_04001*)

[SWS_CM_90206]{DRAFT} **DTLS secure channel for events using unreliable transport** ⌈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.⌋(*RS_SEC_04001*)

[SWS_CM_90207]{DRAFT} **TLS secure channel for fields** ⌈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.⌋(*RS_SEC_04001*)

[SWS_CM_90209]{DRAFT} **IPsec secure channel between communication nodes and Transport of Service communication over an IPsec security association** ⌈An IPsec secure channel shall be created and used according to the requirements and constraints specified in [SWS_CM_90117] and [SWS_CM_90118].⌋(*RS_SEC_04001*)

7.9.2.2.3 Raw Data Streaming

Raw Data Stream communication can be transported via TCP and UDP. Therefore different security mechanism have to be available to secure the stream communication. The following security protocols are currently supported:

- TLS
- DTLS
- IPSec

[SWS_CM_90213] TLS secure channel for raw data streams using reliable transport ⌈A TLS secure channel shall be created and used if

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

⌋(*RS_SEC_04001*)

[SWS_CM_90214] DTLS secure channel for methods using unreliable transport ⌈A DTLS secure channel shall be created and used if:

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

⌋(*RS_SEC_04001*)

[SWS_CM_90215] IPsec secure channel between communication nodes and Transport of Raw Data Stream communication over an IPsec security association
⌈An IPsec secure channel shall be created and used according to the requirements and constraints specified in [SWS_CM_90117] and [SWS_CM_90118], but applying the EthernetRawDataStreamMapping to map to the EthernetCommunicationConnector.⌋(*RS_SEC_04001*)

7.9.2.3 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.

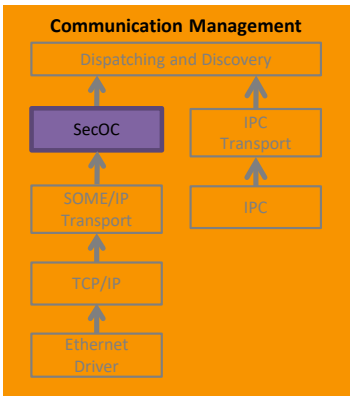


Figure 7.35: SecOC embedded in the Adaptive Communication Management

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 7.36 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 [5]).

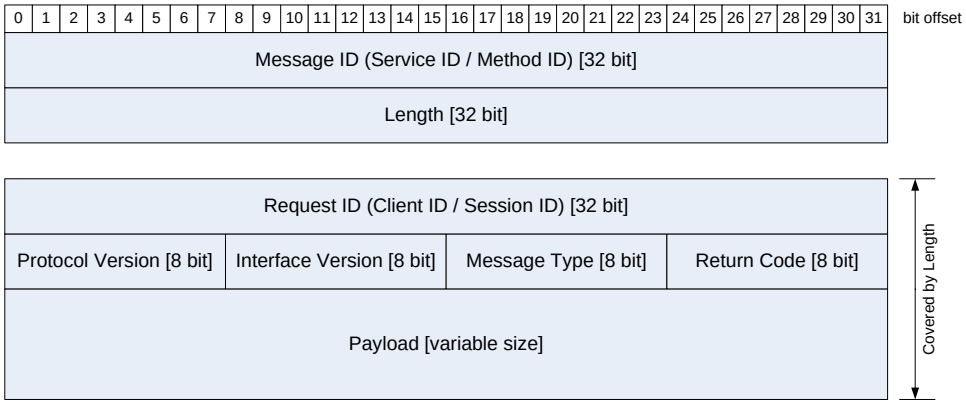


Figure 7.36: SOME/IP header structure

In figure 7.38 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.

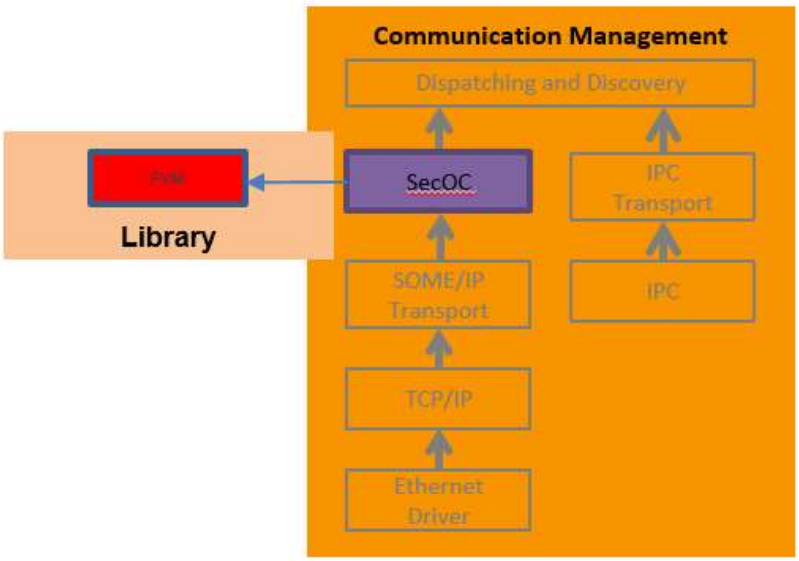


Figure 7.37: Freshness Value Management Pluggable Library

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*.

7.9.2.3.1 SOME/IP network binding

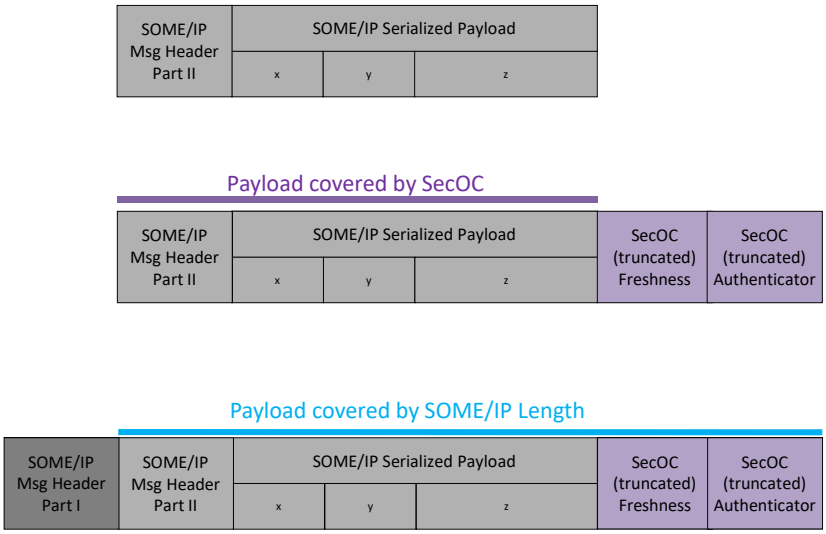


Figure 7.38: Payload covered by SecOC and SOME/IP transport

[SWS_CM_90108]{DRAFT} **SecOC secure channel for methods using reliable transport** ⌈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`.

⌋*(RS_SEC_04001)*

[SWS_CM_90115]{DRAFT} **SecOC secure channel for methods using unreliable transport** ⌈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`.

⌋(*RS_SEC_04001*)

[SWS_CM_90109]{DRAFT} **SecOC secure channel for events and triggers using reliable transport** ⌈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`.

⌋(*RS_SEC_04001*)

[SWS_CM_90116]{DRAFT} **SecOC secure channel for events and triggers using unreliable transport** ⌈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`.

⌋(*RS_SEC_04001*)

[SWS_CM_90110]{DRAFT} **SecOC secure channel for fields** ⌈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`.⌋ (*RS_SEC_04001*)

[SWS_CM_11271]{DRAFT} **SecOC secure channel behavior** ⌈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_11272], [SWS_CM_11274]
- reception according to [SWS_CM_11273], [SWS_CM_11275]

⌋(*RS_SEC_04001*)

[SWS_CM_11272]{DRAFT} **Lifecycle management of FVM** ⌈The lifecycle of an `SecOC FreshnessValueManager` implementation shall be managed by ara::com.⌋ (*RS_SEC_04001*)

[SWS_CM_11273]{DRAFT} **Initialization of the FVM** ⌈

- Initializing of the SecOC FreshnessValueManager implementation by calling Freshness Value Mananement Library API ara::com::secoc::FVM::Initialize.

⌋(*RS_SEC_04001*)

[SWS_CM_11274]{DRAFT} **SecOC secure channel sending** ⌈If a message is configured to be SecOC sent, the message shall be secured according to [27] 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 ara::com::secoc::FVM::GetTxFreshness with the freshnessValueId
- calculate the MAC using the Authentic message ([PRS_SecOc_00200] see [27]), (optionally the Freshness Value), and the dataId
- if the attribute authInfoTxLength is defined, the Authenticator ([PRS_SecOc_00210] see [27]) shall be truncated
- if the attribute freshnessValueTxLength is defined, the Freshness Value shall be truncated ([PRS_SecOc_00201] see [27])
- combine the Authentic message, (truncated) Freshness Value, and (truncated) Authenticator ([PRS_SecOc_00211] see [27])
- 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_00207, PRS_SecOc_00208, PRS_SecOc_00209, PRS_SecOc_00210, PRS_SecOc_00211, PRS_SecOc_00212] (see [27])⌋(*RS_SEC_04001*)

[SWS_CM_11275]{DRAFT} **SecOC secure message build attempts** ⌈For every message to be send and secured with SecOC [27] an Authentication Build Counter([PRS_SecOc_00202] see [27]) shall be maintained:

- the Authentication Build Counter shall be set to 0 if the operation was successful.
- if the query of the freshness value ara::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([PRS_SecOc_00206] see [27]), 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_00201, PRS_SecOc_00202, PRS_SecOc_00203, PRS_SecOc_00204, PRS_SecOc_00205, PRS_SecOc_00206] (see [27])⌋*(RS_SEC_04001)*

[SWS_CM_11276]{DRAFT} **SecOC secure channel reception** ⌈If a message is configured to be SecOC received then the message shall be verified according to [27] 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 ara::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 [27]) from the Secured message, otherwise the Freshness Value([PRS_SecOc_00316] see [27]) shall be extracted from the Secured message itself
- if the attribute authInfoTxLength is defined, the Truncated Authenticator([PRS_SecOc_00315] see [27]) shall be extracted from the Secured message, otherwise the Authenticator([PRS_SecOc_00317] see [27]) 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([PRS_SecOc_00315] 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_00313, PRS_SecOc_00314, PRS_SecOc_00315, PRS_SecOc_00316, PRS_SecOc_00317, PRS_SecOc_00318, PRS_SecOc_00319, PRS_SecOc_00330] (see [27])⌋*(RS_SEC_04001)*

[SWS_CM_11277]{DRAFT} **SecOC secure message verification attempts** ⌈For every message received and secured with SecOc, an Authentication Build Counter([PRS_SecOc_00301] 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` `ara::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 an implementation specific threshold `SecOCAuthVerifyAttempts`([PRS_SecOc_00307] see [27]), 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`([PRS_SecOc_00315] was successful but the verification failed for the `SecOC` implementation specific threshold `SecOCAuthVerifyAttempts`([PRS_SecOc_00306] see [27]), 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_00301, PRS_00302, PRS_00303, PRS_00304, PRS_00305, PRS_00306, PRS_00307, PRS_00308, PRS_00309, PRS_00310, PRS_00311, PRS_00312] (see [27])⌋*(RS_SEC_04001)*

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]{DRAFT} **SecOC verification results** ⌈Communication Management shall make each verification result (`VerificationStatusResult`) accessible via the `VerificationStatus` service.⌋*(RS_SEC_04001)*

[SWS_CM_11279]{DRAFT} **SecOc override the verification result** ⌈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.⌋*(RS_SEC_04001)*

7.9.2.3.2 Signal based network binding

The SOME/IP Message Header as shown in figure 7.36 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 [5]).

In case of signal-service-translation only a partial header is used, namely the Part I. In figure 7.39 the handling of the Header Part I, the signal based payload, and the SecOC part is illustrated.

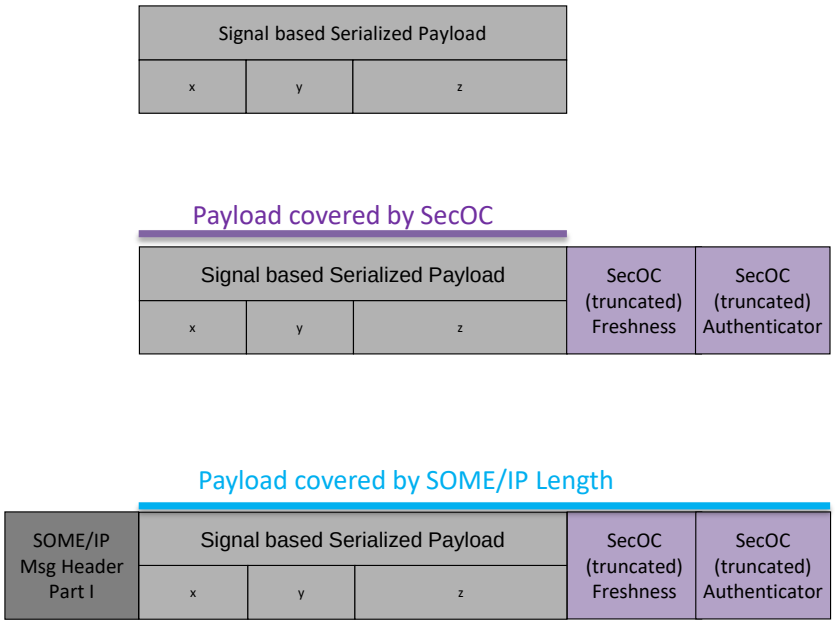


Figure 7.39: Payload covered by SecOC and Signal2Service transport

[SWS_CM_11346]{DRAFT} ⌈If the `ISignalTriggering` is used in a signal-service-translation (the attribute `SomeipEventDeployment.serializer` equals `signal-Based`), CM shall check if the `PduTriggering` of this `ISignalIPdu` is referenced by a `SecuredIPdu` and use the `SecureCommunicationAuthenticationProps`, `SecureCommunicationFreshnessProps` and `SecureCommunicationProps` of the `SecuredIPdu` as configuration of SecOc.⌋*(RS_SEC_04001)*

As described in `Security` chapter of [6], 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 7.40:

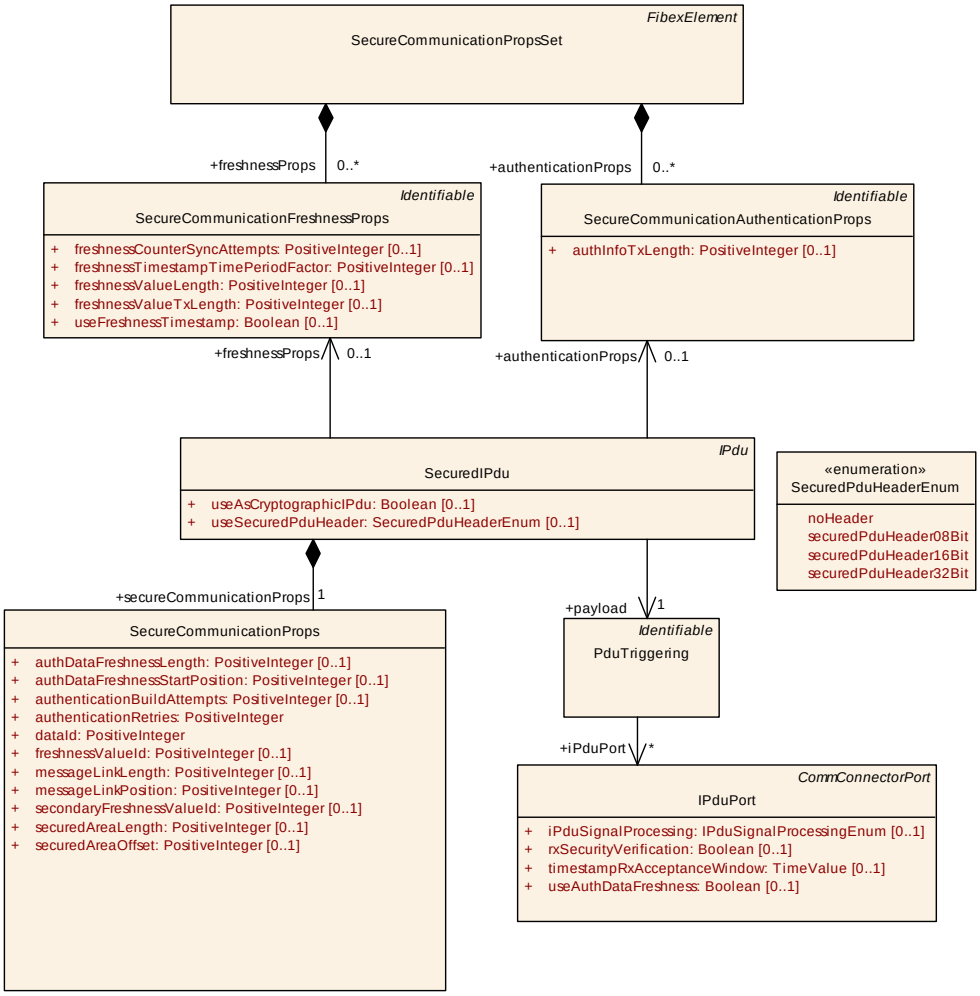


Figure 7.40: Input for for signal based SecOC configuration

7.9.2.4 IPsec

IPsec provides cryptographic protection for IP datagrams in IPv4 and IPv6 network packets.

[SWS_CM_90117]{DRAFT} **IPsec secure channel between communication nodes** ⌈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 `IPSecRule`s in the `IPSecConfig` define security associations between the `NetworkEndpoint` that aggregates this `IPSecConfig` and remote nodes that are defined by the referenced `remoteIpAddress`.⌋*(RS_SEC_04001)*

[SWS_CM_90118]{DRAFT} **Transport of Service communication over an IPsec security association** ⌈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.⌋*(RS_SEC_04001)*

7.9.2.5 DDS Security

DDS Security, as defined in [25], 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 [28].

[SWS_CM_90218]{DRAFT} **Enforcement of IAM grants through DDS Security** ⌈Adaptive Applications providing or requiring Service Interface Instances through the DDS Network Binding shall enforce, when provided, deployed DDS Security policies.⌋ *(RS_IAM_00001, RS_IAM_00002)*

7.10 Communication API

In the following chapter the functional API specification shall be described.

7.10.1 Offer service

For the service offering C++ API reference, see chapter 8.1.3.2.

[SWS_CM_00102]{DRAFT} **Uniqueness of offered service on local machine** ⌈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 duplication (i.e., a service with the same `serviceInstanceId`, `serviceInterfaceId` and `majorVersion` on the same VLAN (e.g.according to [constr_1723] of [6]) is already registered, the requested service offering shall not start, and the function shall return positively after error is logged.⌋*(RS_CM_00200, RS_CM_00101, RS_CM_00108)*

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] Protocol where a service is offered ⌈When a new service is offered by the application, the Communication Management shall check over which protocols this service shall be offered. This information is configured in the class of `ServiceInterfaceDeployment` referencing the offered `ServiceInterface` in the role `serviceInterface`. According of the type of the `ServiceInterfaceDeployment` the Communication Management shall trigger the service offering over respective protocol. 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.⌋*(RS_CM_00101)*

[SWS_CM_00104]{DRAFT} **StopOfferService** ⌈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.⌋*(RS_CM_00101)*

7.10.2 Service skeleton creation

For the service skeleton creation C++ API reference, see chapter 8.1.3.3.

[SWS_CM_10410] `InstanceIdentifier` check during the creation of service skeleton ⌈The Communication Management shall check the value of the `InstanceIdentifier` argument: the identifier shall be unique. If the same `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].⌋*(RS_CM_00101)*

[SWS_CM_10450] `InstanceSpecifier` check during the creation of service skeleton ⌈The Communication Management shall check the value of the `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].⌋*(RS_CM_00101, RS_AP_00137)*

[SWS_CM_10451] `InstanceIdentifierContainer` check during the creation of service skeleton ⌈The Communication Management shall check the value of the `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 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].⌋*(RS_CM_00101)*

7.10.3 Processing of service methods

For the processing of service methods C++ API reference, see chapter 8.1.3.7.

[SWS_CM_10411]{DRAFT} **Service method processing modes** ⌈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.

⌋(*RS_CM_00211*)

7.10.4 Registering get handlers for fields

For the registering get handlers for fields C++ API reference, see chapter 8.1.3.8.

[SWS_CM_10412]{DRAFT} **Invoking GetHandlers** ⌈The registered `GetHandler` shall be called by the implementation whenever the Communication Management receives a `Get`.⌋(*RS_CM_00218*)

7.10.5 Registering set handlers for fields

For the registering set handlers for fields C++ API reference, see chapter 8.1.3.9.

[SWS_CM_10413]{DRAFT} **Invoking SetHandlers** ⌈The registered `SetHandler` shall be called by the implementation whenever the Communication Management receives a `Set`.⌋(*RS_CM_00218*)

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]{DRAFT} **Notify the Field value after a call to the SetHandler function** ⌈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]).⌋(*RS_CM_00218*)

[SWS_CM_00128]{DRAFT} **Ensuring the existence of valid Field values** ⌈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::kFieldValueIsNotValid` shall be returned in the result type of OfferService(). The error shall be logged.⌋(*RS_CM_00218*)

[SWS_CM_00129]{DRAFT} **Ensuring the existence of SetHandler** ⌈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::kSetHandlerNotSet` shall be returned in the result type of OfferService(). The error shall be logged.⌋*(RS_CM_00218)*

7.10.6 Find service

For the find service C++ API reference, see chapter 8.1.3.10.

[SWS_CM_00124]{DRAFT} **Find service handler invocation** ⌈After calling the `StartFindService` method, the `FindServiceHandler` shall be called by the Communication Management software to receive the found services. By the first call, the `FindServiceHandler` shall receive the initially known matches, if there are any. In following, the `FindServiceHandler` shall be called every time the availability of any of the services matching the given instance criteria changes.⌋*(RS_CM_00102)*

[SWS_CM_10382]{DRAFT} **Calling stop find service for already stopped finds** ⌈Calls to the `StopFindService` method using a `FindServiceHandle` obtained from a `StartFindService` that already has been stopped shall be silently ignored.⌋ *(RS_CM_00102)*

7.10.7 Receive events

For the event data access C++ API reference, see chapter 8.1.3.14.

[SWS_CM_00709] FIFO semantics ⌈The Communication Management shall provide buffering with FIFO semantics between sender and receiver of events.⌋*(RS_CM_00203)*

[SWS_CM_00710]{DRAFT} **No implicit context switches** ⌈Reception of a new event shall not lead to an implicit context switch in the receiver process when polling behavior is employed (i.e. No ReceiveHandler has been set via `SetReceiveHandler()`)⌋ *(RS_CM_00203)*

7.10.7.1 Receive event by polling

For the polling access no additional APIs on top of 8.1.3.14 are needed.

7.10.7.2 Receive event by getting triggered

For the receive event by getting triggered C++ API reference, see chapter 8.1.3.15.

[SWS_CM_00182]{DRAFT} **Event Receive Handler call serialization** ⌈The Communication Management shall serialize calls to the registered `EventReceiveHandler` function as it is not guaranteed that the callback function is re-entrant.⌋*(RS_CM_00203)*

[SWS_CM_00711]{DRAFT} ⌈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.⌋*(RS_CM_00203)*

7.10.8 Call a service method

For the call a service method C++ API reference, see chapter 8.1.3.19.

[SWS_CM_10414]{DRAFT} **Initiate a method call** ⌈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.⌋*(RS_CM_00212, RS_CM_00213, RS_AP_00138)*

[SWS_CM_10371]{DRAFT} **Context of return checked errors** ⌈If during processing of a method call one of the checked errors (see subsubsection 8.1.2.6) occurs, the corresponding `ara::core::ErrorCode` shall be returned in the context of the `ara::core::Future::GetResult()`/`ara::core::Future::get()` call.⌋*(RS_CM_00211, RS_CM_00212, RS_CM_00213, RS_CM_00214)*

[SWS_CM_90436]{DRAFT} **No checked errors for `Fire and Forget` method calls** ⌈There shall be no checked errors returned for `Fire and Forget` method calls.⌋*(RS_CM_00225)*

7.10.9 Update notification events for fields

[SWS_CM_00120]{DRAFT} **Provision of an update notification event for a Field** ⌈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.⌋*(RS_CM_00218)*

7.10.10 Instance Specifier Translation

For the instance specifier translation C++ API reference, see chapter 8.1.3.22.

[SWS_CM_10452]{DRAFT} **InstanceSpecifier translation to InstanceIdentifiers** ⌈The Communication Management shall translate an `InstancSpecifier` to `InstanceIdentifier`s. Based on the match there shall be zero, 1 or multiple `InstanceIdentifier`s .⌋*(RS_CM_00200, RS_AP_00137)*

7.10.11 Invalid Value

[SWS_CM_10453]{DRAFT} **Implementation of `invalidValue`** ⌈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

⌋*(RS_CM_00001)*

Note: `invalidValue`s are only applicable to `CppImplementationDataType` of `category` `TYPE_REFERENCE` and `STRING`.

8 Communication API specification

While the primary focus of ara::com is targeted towards an implementation in the C++ programming language, the following chapter structures of the document allow for future versions to add further Language Bindings.

8.1 C++ language binding

8.1.1 API Header files

This chapter specifies those C++ header files used directly in the API implementation of the `ServiceInterface` in an Adaptive Application. As part of the Adaptive Platform Methodology, these C++ header files are generated by a workflow tool directly from the `ServiceInterface` `ARXML` configuration either as part of the i) *Service/Common/-Types Header file* generation, or the ii) *Implementation Types header file* generation [24].

The following requirements are applicable for all header files; requirements which are specific for a header file are described in own sub-chapters.

[SWS_CM_01020]{DRAFT} **Common/Service header files directory structure** ⌈The *Service header file*s defined by [SWS_CM_01002] and the *Common header file*s defined by [SWS_CM_01012] shall be located within the folder:

`<namespace[0]>/<namespace[1]>/.../<namespace[n]>/`

where:
`<namespace[0]>` ... `<namespace[n]>` are the namespace names as defined in [SWS_CM_01005].⌋*(RS_CM_00001, RS_AP_00114)*

[SWS_CM_12000]{DRAFT} **Implementation types header files directory structure** ⌈The communication management expects the *Implementation Types header file*s generated according to [SWS_CM_12001] shall be located within the directory according to [SWS_LBAP_00034].⌋*(RS_CM_00001, RS_AP_00114)*

8.1.1.1 Service header files

The *Service header file*s are the central definition of the ara::com API and any associated data structures that are required by the AdaptiveApplication software components to use the communication management.

[SWS_CM_01002]{DRAFT} **Service header files existence** ⌈The communication management shall provide one *Proxy header file* and one *Skeleton header file* for each `ServiceInterface` defined in the input by using the file name `<name>_proxy.h` for the *Proxy header file* and `<name>_skeleton.h` for the *Skeleton header file*,

where <name> is the ServiceInterface.shortName converted to lower-case letters.⌋(*RS_CM_00001*, *RS_AP_00114*, *RS_AP_00116*)

[SWS_CM_01004]{DRAFT} **Inclusion of common header file** ⌈The *Proxy* and *Skeleton header file* shall include the *Common header file*:

```
#include "<namespace[0]>/<namespace[1]>/.../<namespace[n]>/<name>
    _common.h"
```

where:
<namespace[0]> ... <namespace[n]> are the namespace names as defined in [SWS_CM_01005] and [SWS_LBAP_00035]. <name> is the the ServiceInterface.shortName converted to lower-case letters.⌋(*RS_CM_00001*, *RS_AP_00114*)

Namespaces are used to separate the definition of services from each other to prevent name conflicts and they allow to use reasonably short names.

[SWS_CM_01005]{DRAFT} **Namespace of Service header files** ⌈Based on the symbol attributes of the ordered SymbolProps aggregated by PortInterface in role namespace, the C++ namespace of the *Service header file* shall be:

```
namespace <ServiceInterface.namespace[0].symbol> {
namespace <ServiceInterface.namespace[1].symbol> {
namespace <...> {
namespace <ServiceInterface.namespace[n].symbol> {
...
} // namespace <ServiceInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <ServiceInterface.namespace[1].symbol>
} // namespace <ServiceInterface.namespace[0].symbol>
```

with all namespace names converted to lower-case letters.⌋(*RS_CM_00002*, *RS_AP_00114*)

Note: In order to avoid name clashes between Events, Fields, and Methods of different ServiceInterfaces in situation where the Events (ServiceInterface.event), Fields (ServiceInterface.field), and Methods (ServiceInterface.method) of the different ServiceInterfaces carry the same shortName, it is highly recommend to place different ServiceInterfaces into dedicated unique C++ namespaces. This is achieved by attaching corresponding ordered SymbolProps to the ServiceInterfaces where the ordered SymbolProps differ in at least one of their symbol attributes.

Starting from the innermost namespace as defined by [SWS_CM_01005], there are additional C++ namespaces for the proxy or the skeleton and for the events and methods. These namespaces are used for the declarations and definitions as described in chapter 8.1.3.

[SWS_CM_01006]{DRAFT} **Service skeleton namespace** ⌈The C++ namespace for a specific service skeleton class shall be:

```
namespace skeleton {
...
} // namespace skeleton
```

⌋(*RS_CM_00002*, *RS_AP_00114*)

[SWS_CM_01007]{DRAFT} **Service proxy namespace** ⌈The C++ namespace for a specific service proxy class shall be:

```
namespace proxy {
...
} // namespace proxy
```

⌋(*RS_CM_00002*, *RS_AP_00114*)

[SWS_CM_01009]{DRAFT} **Service events namespace** ⌈The *Proxy* and *Skeleton header file* shall provide a C++ namespace for the definition of events within the namespace defined by [SWS_CM_01006] and [SWS_CM_01007] respectively:

```
namespace events {
...
} // namespace events
```

⌋(*RS_AP_00114*, *RS_CM_00002*)

[SWS_CM_01015]{DRAFT} **Service methods namespace** ⌈The *Proxy* and *Skeleton header file* shall provide a C++ namespace for the definition of methods within the namespace defined by [SWS_CM_01006] and [SWS_CM_01007] respectively:

```
namespace methods {
...
} // namespace methods
```

⌋(*RS_CM_00002*, *RS_AP_00114*)

[SWS_CM_01031]{DRAFT} **Service fields namespace** ⌈The *Proxy* and *Skeleton header file* shall provide a C++ namespace for the definition of fields within the namespace defined by [SWS_CM_01006] and [SWS_CM_01007] respectively:

```
namespace fields {
...
} // namespace fields
```

⌋(*RS_CM_00002*, *RS_CM_00216*, *RS_AP_00114*)

As a summary of the C++ namespace requirements [SWS_CM_01005], [SWS_CM_01006], and [SWS_CM_01009], the namespace hierarchy in the *Skeleton header file* looks like:

```
namespace <ServiceInterface.namespace[0].symbol> {
namespace <ServiceInterface.namespace[1].symbol> {
namespace <...> {
namespace <ServiceInterface.namespace[n].symbol> {
namespace skeleton {

namespace events {
...
} // namespace events

namespace methods {
```

```
...
} // namespace methods

namespace fields {
...
} // namespace fields

...
} // namespace skeleton
} // namespace <ServiceInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <ServiceInterface.namespace[1].symbol>
} // namespace <ServiceInterface.namespace[0].symbol>
```

As a summary of the C++ namespace requirements [SWS_CM_01005], [SWS_CM_01007], [SWS_CM_01009], and [SWS_CM_01015], the namespace hierarchy in the *Proxy header file* looks like:

```
namespace <ServiceInterface.namespace[0].symbol> {
namespace <ServiceInterface.namespace[1].symbol> {
namespace <...> {
namespace <ServiceInterface.namespace[n].symbol> {
namespace proxy {

namespace events {
...
} // namespace events

namespace methods {
...
} // namespace methods

namespace fields {
...
} // namespace fields

...
} // namespace proxy
} // namespace <ServiceInterface.namespace[n].symbol>
} // namespace <...>
} // namespace <ServiceInterface.namespace[1].symbol>
} // namespace <ServiceInterface.namespace[0].symbol>
```

8.1.1.2 Common header file

The *Common header file* includes the ara::com specific type declarations derived from the `ApApplicationError`s composed by a particular `ServiceInterface` as well Service Identifier type declarations related to a particular `ServiceInterface`.

[SWS_CM_01012]{DRAFT} **Common header file existence** ⌈The communication management shall provide a *Common header file* for each `ServiceInterface` defined in the input by using the file name `<name>_common.h`, where `<name>` is the

ServiceInterface.shortName converted to lower-case letters.⌋(*RS_CM_00001*, *RS_AP_00114*, *RS_AP_00116*)

As a minimal requirement, the *Types header file* and the *Implementation Types header files* need to be included.

[SWS_CM_01001]{DRAFT} **Inclusion of Types header file** ⌈The *Common header file* shall include the *Types header file*:

```
#include "ara/com/types.h"
```

⌋(*RS_CM_00001*, *RS_AP_00114*)

[SWS_CM_10372]{DRAFT} **Inclusion of Implementation Types header files** ⌈The *Common header file* shall include the *Implementation Types header files* of those CppImplementationDataTypes that are actually *used* by the particular ServiceInterface:

```
#include "<namespace[0]>/<namespace[1]>/.../<namespace[n]>/impl_type_<
    symbol>.h"
```

where <namespace[0..n]> is the namespace hierarchy defined in [SWS_LBAP_00035], and <symbol> is the Cpp Implementation Data Type symbol according to [24] converted to lower-case letters.⌋(*RS_CM_00001*, *RS_AP_00114*)

It is not mandatory that all declarations and definitions are located directly in the *Common header file*. A Communication Management implementation might also distribute the declarations and definitions into different header files, but at least all those header files need to be included into the *Common header file*.

[SWS_CM_10370]{DRAFT} **Common header file for Application Errors** ⌈The *Common header file* shall include the class definitions for all ApApplicationErrorDomains for the ApApplicationErrors of a ServiceInterface according to [SWS_CM_11266].⌋(*RS_CM_00001*)

[SWS_CM_01017]{DRAFT} **Service Identifier Type definitions in Common header file** ⌈The *Common header file* shall include the information to identify the service type according to the requirement [SWS_CM_01010].⌋(*RS_CM_00001*)

[SWS_CM_01008]{DRAFT} **Namespace for Service Identifier Type definitions** ⌈The declarations and definitions according to [SWS_CM_01017] shall be located in the C++ namespace as defined by [SWS_CM_01005] to match to the namespace of the related skeleton and proxy header file.⌋(*RS_CM_00002*)

8.1.1.3 Types header file

The *Types header file* includes the data type definitions which are specific for the ara::com API. Such data type definitions are used in the standardized proxy and skeleton interfaces defined in chapter 8.1.3.

[SWS_CM_01013]{DRAFT} **Types header file existence** ⌈The communication management shall provide a *Types header file* by using the file name `types.h`.⌋*(RS_CM_00001, RS_AP_00114, RS_AP_00116)*

[SWS_CM_01018]{DRAFT} **Types header file namespace** ⌈The C++ namespace for the data type definitions included by the *Types header file* shall be:

```
namespace ara {
namespace com {
...
} // namespace com
} // namespace ara
```

⌋*(RS_CM_00002, RS_AP_00114)*

It is not mandatory that all data type definitions are located directly in the *Types header file*. A Communication Management implementation might also distribute the definitions into different header files, but at least all those header files need to be included into the *Types header file*.

[SWS_CM_01019]{DRAFT} **Data Type declarations in Types header file** ⌈The *Types header file* shall include the data type definitions according to [SWS_CM_00301], [SWS_CM_00302], [SWS_CM_00303], [SWS_CM_00304], [SWS_CM_00383], [SWS_CM_00306], [SWS_CM_00308], [SWS_CM_00309], [SWS_CM_00310], and [SWS_CM_00311].⌋*(RS_CM_00001)*

8.1.1.4 Implementation Types header files

As part of the Adaptive Application methodology, all referenced `CppImplementationDataType`s in all modeled `ServiceInterface`s for a given Adaptive Application, must be processed by an `ARA generator` to generate the respective C++ language binding representation [29].

The processing rules which specify how an `ARA generator` shall create the *Implementation Types header files* are specified in detail in [24]. The role of communication management is as a 'consumer' of the respective generated *Implementation Types header files*.

[SWS_CM_12001]{DRAFT} **C++ Implementation Data Types files** ⌈The communication management shall use the generated *C++ Implementation Types header file*s produced according to:

- [SWS_LBAP_00032] (header file generation),
- [SWS_LBAP_00033] (header file naming),
- [SWS_LBAP_00034] (directory naming),

⌋*()*

8.1.1.5 Raw Data Stream header file

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

[SWS_CM_10488] Raw data stream header file existence ⌈The communication management shall provide a *Raw data stream header file* by using the file name `raw_data_stream.h`.⌋*(RS_CM_00001)*

[SWS_CM_10489] Raw data stream header file namespace ⌈The C++ namespace for the data type definitions included by the *Raw data stream header file* shall be:

```
namespace ara {
namespace com {
namespace raw {
...
} // namespace raw
} // namespace com
} // namespace ara
```

⌋*(RS_CM_00002)*

[SWS_CM_10490] Data Type declarations in Raw data stream header file ⌈The *Raw data stream header file* shall include the class definitions according to [SWS_CM_10481], [SWS_CM_10482], [SWS_CM_10483], [SWS_CM_10484], [SWS_CM_10485], [SWS_CM_10486] and [SWS_CM_10487].⌋*(RS_CM_00001)*

8.1.2 API Data Types

This chapter describes 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.

8.1.2.1 Service Identifier Data Types

The data types described in this chapter are derived from the ara::com API design and as a part of the API, they are used to identify a specific service or service instance.

A service can be identified at least by a fully qualified name and a version. The `serviceIdentifier` is not visible in the ara::com API, as the specific service skeleton and proxy class itself represent the service type, but the `serviceIdentifier` is needed for the implementation of the Communication Management software. It is defined here to guarantee a minimum amount of information.

[SWS_CM_01010]{DRAFT} **Service Identifier and Service Contract Version** ⌈The Communication Management shall provide a C++ class named `ServiceInterface.shortName`. The class contains at least a fully qualified name identifier (`serviceIdentifier`) , a service contract major (`serviceContractVersionMajor`) and minor (`serviceContractVersionMinor`) version number.

The value of `serviceContractVersionMajor` shall be derived from the `majorVersion` attribute in the `ServiceInterface`. The value of `serviceContractVersionMinor` shall be derived from the `minorVersion` attribute in the `ServiceInterface`.

The exact type of `serviceIdentifier` is specific to the Communication Management software provider. Its concrete realization is implementation defined. To allow for logging and for storing and managing in C++ container classes by the using application, however, the type of the class shall satisfy the `EqualityComparable` requirements according to table 17, the `LessThanComparable` requirements according to table 18, and the `CopyAssignable` requirements according to table 23 of section 17.6.3.1 of [30]. These requirements are fulfilled if the operators `operator==`, `operator<`, and `operator=` as well as a `toString()` method is provided.

```
class <ServiceInterface.shortName> {
 public:
  static const serviceIdentifierType serviceIdentifier;

  static std::uint32_t serviceContractVersionMajor;
  static std::uint32_t serviceContractVersionMinor;
};

class serviceIdentifierType {
  bool operator==(const serviceIdentifierType& other) const;
  bool operator<(const serviceIdentifierType& other) const;
  serviceIdentifierType& operator=(const serviceIdentifierType& other);
```

```
13    ara::core::StringView ToString() const;
14 };
```

⌋(*RS_CM_00200*, *RS_CM_00500*)

There might exist different instances of exactly the same service in the system. To handle this, an `InstanceIdentifier` or an `InstanceSpecifier` are used to identify a specific instance of a service. These are a necessary parameter of the API defined for both the skeleton and proxy side:

- on service skeleton side, it types the parameter needed to identify the service instance when creating an instance by [SWS_CM_00130],[SWS_CM_00152],[SWS_CM_00153].
- on service proxy side, it types the parameter needed to identify the service instance when searching for a specific instance by [SWS_CM_00122] or [SWS_CM_00123].

[SWS_CM_00302]{DRAFT} **Instance Identifier Class** ⌈The Communication Management shall provide a class `InstanceIdentifier`. It contains instance information and information about the service type. This will make the InstanceIdentifier unique for different instances.
The definition of the `InstanceIdentifier` can be extended by the Communication Management software provider, but at least the given Named Constructor `Create()`, the class constructor and the class method signatures must be preserved. `InstanceIdentifier` shall further satisfy the `EqualityComparable` requirements according to table 17, the `LessThanComparable` requirements according to table 18, and the `CopyAssignable` requirements according to table 23 of section 17.6.3.1 of [30] to allow for logging of `InstanceIdentifier`s as well as storing and managing `InstanceIdentifier`s in C++ container classes by the using application. These requirements are fulfilled if the operators `operator==`, `operator<`, and `operator=` as well as a `ToString()` method is provided.

The format of the string passed to the constructor, or returned by the `ToString()` method is specific to the Communication Management software provider, and not standardized.

In case the format of the string representation provided to `Create()` is corrupted, or not compliant to the software provider specification, an error code `ComErrc::kInvalidInstanceIdentifierString` shall be returned in the Result type.
The class constructor shall throw a `ComException` in case the format of the string provided is corrupted, or not compliant to the software provider specification.

```
1 class InstanceIdentifier {
2  public:
3
4   static ara::core::Result<InstanceIdentifier> Create(StringView
      serializedFormat) noexcept;
```

```
  explicit InstanceIdentifier( ara::core::StringView serializedFormat);
  ara::core::StringView ToString() const;
  bool operator==(const InstanceIdentifier& other) const;
  bool operator<(const InstanceIdentifier& other) const;
  InstanceIdentifier& operator=(const InstanceIdentifier& other);
};
```

⌋(*RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00122, RS_AP_00127*)

[SWS_CM_00319]{DRAFT} **Instance Identifier Container Class** ⌈The Communication Management shall provide the definition of a `InstanceIdentifierContainer`. The container holds a list of `InstanceIdentifier`. The assigned data type is allowed to be changed by the Communication Management software provider, but must adhere to the *general container requirements* according to table 96 of section 23.2.1 and the *sequence container requirements* according to table 100 of section 23.2.3 of [30]. A `ara::core::Vector` for example fulfills these requirements.

```
using InstanceIdentifierContainer = ara::core::Vector<InstanceIdentifier>;
```

⌋(*RS_CM_00101, RS_CM_00102, RS_AP_00114, RS_AP_00122*)

The following data types are used for the handling of services on the service consumer side, therefore they are part of the API defined for the proxy side.

To identify a triggered request to find a service, the `StartFindService` method of [SWS_CM_00123] returns a `FindServiceHandle` which is used as parameter to cancel this request with `StopFindService` as described in [SWS_CM_00125].

[SWS_CM_00303]{DRAFT} **Find Service Handle** ⌈The Communication Management shall provide the definition of an opaque `FindServiceHandle` with exactly this name. `FindServiceHandle` shall satisfy the `EqualityComparable` requirements according to table 17, the `LessThanComparable` requirements according to table 18, and the `CopyAssignable` requirements according to table 23 of section 17.6.3.1 of [30] to allow storing and managing `FindServiceHandle`s in C++ container classes by the using application. These requirements are fulfilled if the following operators are provided: `operator==`, `operator<`, and `operator=`. The exact definition of `FindServiceHandle` is communication management implementation specific.⌋(*RS_CM_00102, RS_AP_00122*)

For example, a definition of `FindServiceHandle` could look like this:

```
struct FindServiceHandle {
  internal::ServiceId serviceIdentifier;
  internal::InstanceId instanceIdentifier;
  std::uint32_t uid;

  bool operator==(const FindServiceHandle& other) const;
  bool operator<(const FindServiceHandle& other) const;
  FindServiceHandle& operator=(const FindServiceHandle& other);
  ...
};
```

The usage of the API to find service instances, as defined in [SWS_CM_00122] and [SWS_CM_00123], provides a *handle container* 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.

[SWS_CM_00312]{DRAFT} **Handle Type Class** ⌈The Communication Management shall provide the definition of `HandleType`. It types the handle for a specific service instance and shall contain the information that is needed to create a `ServiceProxy`. The definition of the `HandleType` can be extended by the Communication Management software provider, but at least the given class and class method signatures must be preserved.
`HandleType` shall satisfy the `EqualityComparable` requirements according to table 17 and the `LessThanComparable` requirements according to table 18 of section 17.6.3.1 of [30]. These requirements are fulfilled if the following operators are provided: `operator==` and `operator<`. This, together with [SWS_CM_00317] and [SWS_CM_00318], allows storing and managing `HandleType`s in C++ container classes by the using application.
The definition of the `HandleType` class shall be located inside the `ServiceProxy` class defined by [SWS_CM_00004]. This allows the Communication Management software to provide handles with different implementation dependent on the binding to the represented service.

```
class HandleType {
 public:
  bool operator==(const HandleType& other) const;
  bool operator<(const HandleType& other) const;
  const ara::com::InstanceIdentifier& GetInstanceId() const;
};
```

⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00122)*

Since the Communication Management software is responsible for creation of handles and the application just uses instances of it, the constructor signature is not part of the `HandleType` specification.

[SWS_CM_00317]{DRAFT} **Copy semantics of handle Type Class** ⌈The Communication Management shall provide the possibility to copy construct and copy assign a `HandleType` instance from another instance.

```
HandleType(const HandleType&);
HandleType& operator=(const HandleType&);
```

⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00145)*

[SWS_CM_00318]{DRAFT} **Move semantics of handle Type Class** ⌈The Communication Management shall provide the possibility to move construct and move assign a `HandleType` instance from another instance.

```
HandleType(HandleType &&);
HandleType& operator=(HandleType &&);
```

⌋(*RS_CM_00102*, *RS_AP_00114*, *RS_AP_00145*)

[SWS_CM_11371]{DRAFT} **HandleType destructor** ⌈The Communication Management shall provide a destructor for the the `HandleType`.

```
~HandleType() noexcept;
```

⌋(*RS_AP_00114*, *RS_AP_00145*, *RS_AP_00132*)

[SWS_CM_00304]{DRAFT} **Service Handle Container** ⌈The Communication Management shall provide the definition of a `ServiceHandleContainer`. The container holds a list of service handles and is used as a return value of the `FindService` methods. The assigned data type is allowed to be changed by the Communication Management software provider, but must adhere to the *general container requirements* according to table 96 of section 23.2.1 and the *sequence container requirements* according to table 100 of section 23.2.3 of [30]. A `ara::core::Vector` for example fulfills these requirements.

```
template <typename T>
using ServiceHandleContainer = ara::core::Vector<T>;
```

⌋(*RS_CM_00102*, *RS_AP_00114*, *RS_AP_00122*)

The possibility to continuously find services by registering a *handler function* as defined in [SWS_CM_00123] requires a definition of such a *handler function*. The function implementation itself is to be provided by the proxy user.

[SWS_CM_00383]{DRAFT} **Find Service Handler** ⌈The Communication Management shall provide the definition of `FindServiceHandler` as a function wrapper for the handler function that gets called by the Communication Management software in case the service availability changes. It takes as input parameter a handle container containing handles for all matching service instances and a `FindServiceHandle` which can be used to invoke `StopFindService` (see [SWS_CM_00125]) from within the `FindServiceHandler`.

```
template <typename T>
using FindServiceHandler =
std::function<void(ServiceHandleContainer<T>, FindServiceHandle)>;
```

⌋(*RS_CM_00102*, *RS_AP_00114*, *RS_AP_00120*)

See [SWS_CM_00304] for the type definition of `ServiceHandleContainer`.

8.1.2.2 Event Related Data Types

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* is an alias for an event data type pointer.

`SamplePtr` behaves similar to `std::unique_ptr` but it may be implemented with a subset of features. It also contains an additional method `GetProfileCheckStatus` to access the E2E results provided by `ProfileCheckStatus` of the referred sample.

[SWS_CM_00306]{DRAFT} **Sample Pointer** ⌈The Communication Management shall provide a `SamplePtr` template which provides a pointer to a managed data object. The implementation shall at least contain the following constructors, assign operators and methods:

```
template< typename T >
class SamplePtr {

  // Default constructor
  constexpr SamplePtr() noexcept;

  // semantically equivalent to Default constructor
  constexpr SamplePtr(std::nullptr_t) noexcept;

  // Copy constructor is deleted
  SamplePtr ( const SamplePtr& ) = delete;

  // Move constructor
  SamplePtr( SamplePtr&& ) noexcept;

  // Destructor
  ~SamplePtr() noexcept;

  // Default copy assignment operator is deleted
  SamplePtr& operator=( const SamplePtr& ) = delete;

  // Assignment of nullptr_t
  SamplePtr& operator=(std::nullptr_t) noexcept;

  // Move assignment operator
  SamplePtr& operator=( SamplePtr&& ) noexcept;

  // Dereferences the stored pointer
  T& operator*() const noexcept;
  T* operator->() const noexcept;

  //Checks if the stored pointer is null
  explicit operator bool () const noexept;
```

```
  // Swaps the managed object
  void Swap ( SamplePtr& ) noexcept;

  //Replaces the managed object
  void Reset (std::nullptr_t)  noexcept;

  //Returns the stored object
  T* Get () const noexcept;

  // Returns the end 2 end protection check result
  ara::com::e2e::ProfileCheckStatus GetProfileCheckStatus() const noexcept;

};
```

⌋(*RS_CM_00202*, *RS_CM_00203*, *RS_AP_00114*, *RS_AP_00122*, *RS_AP_00132*, *RS_AP_00135*, *RS_AP_00145*)

[SWS_CM_90420]{DRAFT} **E2E ProfileCheckStatus of a sample** ⌈The `SamplePtr` shall provide the access to the `ProfileCheckStatus` of each sample by means of the method `GetProfileCheckStatus`:

```
ara::com::e2e::ProfileCheckStatus GetProfileCheckStatus() const noexcept;
```

⌋(*RS_E2E_08534*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00132*)

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 *Sample Allocatee Pointer* is an alias for an event data type pointer used both for allocation and data sending.

[SWS_CM_00308]{DRAFT} **Sample Allocatee Pointer** ⌈The Communication Management shall provide the definition of `SampleAllocateePtr` as a pointer to a data sample allocated by the Communication Management implementation. The implementation is allowed to be changed by the Communication Management software provider.

```
template <typename T>
using SampleAllocateePtr = std::unique_ptr<T>;
```

⌋(*RS_CM_00201*, *RS_AP_00114*, *RS_AP_00122*, *RS_AP_00135*)

The event receiver can register an *Event Receive Handler* 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].

[SWS_CM_00309]{DRAFT} **Event Receive Handler** ⌈The Communication Management shall provide the definition of `EventReceiveHandler` as a function wrapper without parameters for the handler function that gets called by the Communication Management software in case new event data arrives for an event. The event receiver must provide the function implementation which is not required to be re-entrant.

The symbolic name is set; for the alias it is recommended to use the C++ general-purpose polymorphic function wrapper `std::function`, but this is not mandatory and is allowed to be changed by the Communication Management software provider.

```
using EventReceiveHandler = std::function<void()>;
```

⌋(*RS_AP_00114*, *RS_CM_00203*, *RS_AP_00120*)

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_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.

[SWS_CM_00310]{DRAFT} **Subscription State** ⌈The Communication Management shall provide an enumeration `SubscriptionState` which defines the subscription state of an event.

```
enum class SubscriptionState : std::uint8_t {
  kSubscribed,
  kNotSubscribed,
  kSubscriptionPending
};
```

⌋(*RS_CM_00103*, *RS_CM_00104*, *RS_CM_00106*, *RS_AP_00114*, *RS_AP_00125*, *RS_AP_00119*)

[SWS_CM_00311]{DRAFT} **Subscription State Changed Handler** ⌈The Communication Management shall provide the definition of `SubscriptionStateChangeHandler` as a function wrapper for the handler function that gets called by the Communication Management software in case the subscription state of an event has changed.

```
using SubscriptionStateChangeHandler =
std::function<void(SubscriptionState)>;
```

⌋(*RS_CM_00103*, *RS_CM_00104*, *RS_CM_00106*, *RS_AP_00114*, *RS_AP_00120*)

8.1.2.3 Trigger Related Data Types

The trigger receiver can register an *Trigger Receive Handler* 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_00249].

[SWS_CM_00351]{DRAFT} **Trigger Receive Handler** ⌈The definition of *Trigger Receive Handler* is the same as *Trigger Receive Handler* defined in [SWS_CM_00309]

```
using TriggerReceiveHandler = std::function<void()>;
```

⌋(*RS_AP_00114, RS_CM_00203, RS_AP_00120*)

The trigger receiver can monitor the state of a service event subscription by requesting or getting a notification of the *Subscription State* (see [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.

8.1.2.4 Method Related Data Types

Service method invocation on provider side can be executed in different processing modes, where the *Method Call Processing Mode* is set as a parameter of the `ServiceSkeleton` constructor defined by [SWS_CM_00130].

[SWS_CM_00301]{DRAFT} **Method Call Processing Mode** ⌈The Communication Management shall provide an enumeration `MethodCallProcessingMode` which defines the processing modes for the service implementation side.

```
enum class MethodCallProcessingMode : std::uint8_t {
  kPoll,
  kEvent,
  kEventSingleThread
};
```

⌋(*RS_CM_00211, RS_AP_00114, RS_AP_00125*)

The expected behavior of each processing mode is described in [SWS_CM_00198].

8.1.2.5 Generic Data Types

8.1.2.5.1 `Future` and `Promise`

The `Future` and `Promise` class templates are described in [16].

8.1.2.5.2 Optional Data Types

The `Optional` class template `ara::core::Optional` used in ara::com to provide access to optional record elements of a `Structure Cpp Implementation Data Type` is described in [16].

8.1.2.5.3 Variant Data Types

The class template `ara::core::Variant` is used to provide a type-save union representation is described in [16]. Whenever there is a mention of the standard C++17 Item `std::variant`, the implied source material is [31].

The class template std::variant at a given time either holds a value of one of its alternative types, or in the case of an error, no value.

[SWS_CM_01050]{DRAFT} **`Variant` Class Template** ⌈The Communication Management shall at least provide an `Variant` class template which provides a type-save union representation.

```
template< class... Types >
class Variant {

  // Default constructor
  Variant() noexcept;
  // Move constructor
  Variant( Variant&& ) noexcept;
  // Copy constructor
  Variant( const Variant& );

  // Converting constructor
  template< class T >
  Variant ( T&& ) noexcept;
  // Explicit converting constructors
  template< class T, class... Args >
  explicit Variant ( std::in_place_type_t<T> , Arg&&... );
  template< class T, class U, class... Args >
  explicit Variant ( std::in_place_type_t<T> , std::initializer_list<U> ,
                     Arg&&... );
  template< std::size_t I, class... Args >
  explicit Variant ( std::in_place_index_t<I> , Arg&&... );
  template< std::size_t I, class U, class... Args >
  explicit Variant ( std::in_place_index_t<I> , std::initializer_list<U> ,
                     Arg&&... );

  // Destructor
  ~Variant() noexcept;

  // Move assignment operator
  Variant& operator=( Variant&& ) noexcept;
  // Default copy assignment operator
  Variant& operator=( const Variant& );
  // Converting assignment operator
  template < class T >
  Variant& operator=( T&& ) noexcept;

  // Returns the zero-based index of the alternative
```

```
  std::size_t index();
  // Checks if the Variant is an invalid state
  bool valueless_by_exception() const noexcept;

  // Modifiers
  template < class T, class... Args >
  void emplace( Args&&... );
  template < class T, class U, class... Args >
  void emplace( std::initializer_list<U> , Args&&... );
  template < std::size_t I, class... Args >
  void emplace( Args&&... );
  template <std::size_t I, class U, class... Args>
  void emplace( initializer_list<U> , Args&&... );

  // Swap
  void swap( Variant& ) noexcept;

};
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00122*, *RS_AP_00132*, *RS_AP_00127*, *RS_AP_00134*, *RS_AP_00145*)

[SWS_CM_01051]{DRAFT} **`Variant` default constructor** ⌈The `Variant` constructor

```
Variant();
```

behaves as the `std::variant` constructor

```
variant();
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00145*)

[SWS_CM_01052]{DRAFT} **`Variant` move constructor** ⌈The `Variant` move constructor

```
Variant( Variant&&) noexcept;
```

behaves as the `std::variant` move constructor

```
constexpr variant( variant&& other ) noexcept;
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00132*, *RS_AP_00145*)

[SWS_CM_01053]{DRAFT} **`Variant` copy constructor** ⌈The `Variant` copy constructor

```
Variant( const Variant& );
```

behaves as the `std::variant` copy constructor

```
constexpr variant( const variant& other );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00145*)

[SWS_CM_01054]{DRAFT} **`Variant` converting constructor** ⌈The `Variant` converting constructor

```
template< class T >
Variant ( T&& ) noexcept;
```

behaves as the `std::variant` converting constructor

```
template< class T >
constexpr variant( TT& t ) noexcept;
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00132*, *RS_AP_00145*)

[SWS_CM_01055]{DRAFT} **`Variant` explicit converting constructor with specified alternative** ⌈The `Variant` explicit converting constructor with specified alternative

```
template< class T, class... Args >
explicit Variant ( std::in_place_type_t<T> , Arg&&... );
```

behaves as the `std::variant` explicit converting constructor with specified alternative

```
template< class T, class... Args >
constexpr explicit variant ( std::in_place_type_t<T> , Arg&&... args );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00132*, *RS_AP_00145*)

[SWS_CM_01056]{DRAFT} **`Variant` explicit converting constructor with specified alternative and initializer list** ⌈The `Variant` explicit converting constructor with specified alternative and initializer list

```
template< class T, class U, class... Args >
explicit Variant ( std::in_place_type_t<T> , std::initializer_list<U> ,
    Arg&&... );
```

behaves as the `std::variant` explicit converting constructor with specified alternative and initializer list

```
template< class T, class U, class... Args >
constexpr explicit variant ( std::in_place_type_t<T> , std::
    initializer_list<U> il, Arg&&... args );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00145*)

[SWS_CM_01057]{DRAFT} **`Variant` explicit converting constructor with alternative specified by index** ⌈The `Variant` explicit converting constructor with alternative specified by index

```
template< std::size_t I, class... Args >
explicit Variant ( std::in_place_index_t<I> , Arg&&... );
```

behaves as the `std::variant` with alternative specified by index

```
template< std::size_t I, class... Args >
constexpr explicit variant ( std::in_place_index_t<I> , Arg&&... args )
    ;
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00145*)

[SWS_CM_01058]{DRAFT} **`Variant` explicit converting constructor with alternative specified by index and initializer list** ⌈The `Variant` explicit converting constructor with alternative specified by index and initializer list

```
template< std::size_t I, class U, class... Args >
explicit Variant ( std::in_place_index_t<I> , std::initializer_list<U>
    , Arg&&... );
```

behaves as the `std::variant` with alternative specified by index and initializer list

```
template< std::size_t I, class U, class... Args >
constexpr explicit variant ( std::in_place_index_t<I> , std::
    initializer_list<U> il, Arg&&... args );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00145*)

[SWS_CM_01059]{DRAFT} **`Variant` destructor** ⌈The `Variant` destructor

```
~Variant() noexcept;
```

behaves as the `std::variant` destructor with noexcept specifier

```
~variant() noexcept;
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00134*, *RS_AP_00145*)

[SWS_CM_01060]{DRAFT} **`Variant` move assignment operator** ⌈The `Variant` move assignment operator

```
Variant& operator=( Variant&& ) noexcept;
```

behaves as the `std::variant` move assignment operator

```
constexpr variant( variant&& rhs ) noexcept
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00132*, *RS_AP_00145*)

[SWS_CM_01061]{DRAFT} **`Variant` default copy assignment operator** ⌈The `Variant` default copy assignment operator

```
Variant& operator=(const Variant&);
```

behaves as the `std::variant` default copy assignment operator

```
variant& operator=( const variant& rhs );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00145*)

[SWS_CM_01062]{DRAFT} **`Variant` converting assignment operator** ⌈The `Variant` converting assignment operator

```
template < class T >
Variant& operator=( T&& ) noexcept;
```

behaves as the `std::variant` converting assignment operator

```
template < class T >
variant& operator=( T&& t ) noexcept;
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00132*)

[SWS_CM_01063]{DRAFT} **`Variant` function to return the zero-based index of the alternative** ⌈The `Variant` function returns the zero-based index of the alternative

```
std::size_t index();
```

behaves as the `std::variant` function to return the zero-based index of the alternative

```
constexpr std::size_t index();
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*)

[SWS_CM_01064]{DRAFT} **`Variant` function to check if the Variant is in invalid state** ⌈The `Variant` function checks if the Variant is in invalid state

```
bool valueless_by_exception() const noexcept;
```

behaves as the `std::variant` function to return false if the variant holds a value, else true

```
constexpr bool valueless_by_exception() const noexcept;
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00132*)

[SWS_CM_01066]{DRAFT} **`Variant` function to create a new value in-place, in an existing Variant object** ⌈The `Variant` creates a new value in-place, in an existing Variant object

```
template < class T, class... Args >
void emplace( Args&&... );
```

behaves as the `std::variant` emplace function to create a new value in-place, in an existing Variant object

```
template < class T, class... Args >
void emplace( Args&&... args );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*)

[SWS_CM_01067]{DRAFT} **`Variant` function to create a new value in-place, in an existing Variant object using an initializer list** ⌈The `Variant` creates a new value in-place, in an existing Variant object using initializer list

```
template < class T, class U, class... Args >
void emplace( std::initializer_list<U> , Args&&... );
```

behaves as the `std::variant` emplace function to create a new value in-place, in an existing Variant object using an initializer list

```
template < class T, class U, class... Args >
void emplace( std::initializer_list<U> il , Args&&... args );
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*)

[SWS_CM_01068]{DRAFT} **`Variant` function to create a new value in-place, in an existing Variant object by destoying and initializing the contained value** ⌈The `Variant` creates a new value in-place, in an existing Variant object by destroying and initializing the contained value

```
template < std::size_t I, class... Args >
void emplace( Args&&... );
```

behaves as the `std::variant` emplace function to create a new value in-place, in an existing Variant object by destroying and initializing the contained value

```
template < std::size_t I, class... Args >
void emplace( Args&&... args );
```

The behavior is undefined if I is not less than sizeof...(Types)⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*)

[SWS_CM_01069]{DRAFT} **`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** ⌈The `Variant` creates a new value in-place, in an existing Variant object by destroying and initializing the contained value using an initializer list

```
template <std::size_t I, class U, class... Args>
void emplace( initializer_list<U> , Args&&... );
```

behaves as the `std::variant` emplace function to create a new value in-place, in an existing Variant object by destroying and initializing the contained value using an initializer list

```
template <std::size_t I, class U, class... Args>
void emplace( initializer_list<U> il , Args&&... args );
```

The behavior is undefined if I is not less than sizeof...(Types)⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*)

[SWS_CM_01065]{DRAFT} **`Variant` function to swap two Variants** ⌈The `Variant` function swaps two Variants

```
void swap( Variant& ) noexcept;
```

behaves as the `std::variant` function to swap two Variants

```
void swap( Variant& rhs ) noexcept;
```

⌋(*RS_CM_00205*, *RS_SOMEIP_00050*, *RS_AP_00114*, *RS_AP_00132*)

8.1.2.6 Error Types

[SWS_CM_11265]{DRAFT} **Use of general ara::com errors** ⌈Any `Checked Error` of a service interface shall be reported via the return type as specified in [16].⌋*(RS_CM_00211, RS_AP_00119)*

In `ara::com`, there are the following types of Checked Errors:

1. General ara::com errors: These errors can occur in a call of a service interface method but are not specific to a certain service interface. 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::E2EErrorDomain` (see chapter 8.1.2.7)
3. Application Errors: These errors are specific to a certain service interface call. They are defined as `ApApplicationError` in the meta-model.
4. Communication Group Errors: These errors are specific to communication groups. They are defines in the error domain `ara::com::cg::CgErrorDomain`

Errors can also occur in a call to a `RawDataStreamClientInterface` or `RawDataStreamServerInterface` instance method. These errors are defined in the error domain `ara::com::raw::RawErrorDomain`

[SWS_CM_11264]{DRAFT} **Definition general ara::com errors** ⌈General ara::com errors shall be defined in the error domain `ara::com::ComErrorDomain` in accordance with [16].⌋*(RS_CM_00102, RS_AP_00115, RS_AP_00119)*

[SWS_CM_11267]{DRAFT} **General errors domain** ⌈Error domain to describe general ara::com errors `ara::com::ComErrorDomain` shall be defined. It shall have the shortname `Com` and the identifier 0x8000'0000'0000'1267.⌋*(RS_AP_00130)*

[SWS_CM_10432]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	ComErrc	
Scope:	namespace ara::com	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class ComErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kServiceNotAvailable= 1	Service is not available.
	kMaxSamplesExceeded= 2	Application holds more SamplePtrs than commited in Subscribe().
	kNetworkBindingFailure= 3	Local failure has been detected by the network binding.
	kGrantEnforcementError= 4	Request was refused by Grant enforcement layer.
	kPeerIsUnreachable= 5	TLS handshake fail.
	kFieldValueIsNotValid= 6	Field Value is not valid,.
	kSetHandlerNotSet= 7	SetHandler has not been registered.

▽

△

	kUnsetFailure= 8	Failure has been detected by unset operation.
	kSampleAllocationFailure= 9	Not Sufficient memory resources can be allocated.
	kIllegalUseOfAllocate= 10	The allocation was illegally done via custom allocator (i.e., not via shared memory allocation).
	kServiceNotOffered= 11	Service not offered.
	kCommunicationLinkError= 12	Communication link is broken.
	kCommunicationStackError= 14	Communication Stack Error, e.g. network stack, network binding, or communication framework reports an error
	kInstanceIDCouldNotBeResolved= 15	ResolveInstanceIDs() failed to resolve InstanceID from InstanceSpecifier, i.e. is not mapped correctly.
	kMaxSampleCountNotRealizable= 16	Provided maxSampleCount not realizable.
	kWrongMethodCallProcessingMode= 17	Wrong processing mode passed to constructor method call.
	kErroneousFileHandle= 18	The FileHandle returned from FindServce is corrupt/ service not available.
	kCouldNotExecute= 19	Command could not be executed in provided Execution Context.
	kInvalidInstanceIdentifierString= 20	Given InstanceIdentifier string is corrupted or non-compliant.
Header file:	#include "ara/com/com_error_domain.h"	
Description:	The ComErrc enumeration defines the error codes for the ComErrorDomain. .	

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

[SWS_CM_11327]{DRAFT} ⌈

Kind:	class
Symbol:	ComException
Scope:	namespace ara::com
Base class:	ara::core::Exception
Syntax:	`class ComException : public Exception {...};`
Header file:	#include "ara/com/com_error_domain.h"
Description:	Defines a class for exceptions to be thrown by the Communication APIs. .

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

[SWS_CM_11328]{DRAFT} ⌈

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

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

[SWS_CM_11329]{DRAFT} ⌈

Kind:	class
Symbol:	ComErrorDomain
Scope:	namespace ara::com
Base class:	ara::core::ErrorDomain
Syntax:	`class ComErrorDomain final : public ErrorDomain {...};`
Unique ID:	0x8000'0000'0000'1267
Header file:	#include "ara/com/com_error_domain.h"
Description:	Defines a class representing the Communication error domain.

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

[SWS_CM_11336]{DRAFT} ⌈

Kind:	type alias
Symbol:	Errc
Scope:	class ara::com::ComErrorDomain
Derived from:	ComErrc
Syntax:	`using Errc = ComErrc;`
Header file:	#include "ara/com/com_error_domain.h"
Description:	Alias for the error code value enumeration.

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

[SWS_CM_11337]{DRAFT} ⌈

Kind:	type alias
Symbol:	Exception
Scope:	class ara::com::ComErrorDomain
Derived from:	ComException
Syntax:	`using Exception = ComException;`
Header file:	#include "ara/com/com_error_domain.h"
Description:	Alias for the exception base class.

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

[SWS_CM_11330]{DRAFT} ⌈

Kind:	function
Symbol:	ComErrorDomain()
Scope:	class ara::com::ComErrorDomain
Syntax:	`constexpr ComErrorDomain () noexcept;`
Exception Safety:	noexcept
Header file:	#include "ara/com/com_error_domain.h"
Description:	Constructs a new ComErrorDomain object.

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

[SWS_CM_11331]{DRAFT} ⌈

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

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

[SWS_CM_11332]{DRAFT} ⌈

Kind:	function	
Symbol:	Message(CodeType errorCode)	
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 error code.
Exception Safety:	noexcept	
Header file:	#include "ara/com/com_error_domain.h"	
Description:	Returns the message associated with errorCode.	

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

[SWS_CM_11333]{DRAFT} ⌈

Kind:	function	
Symbol:	ThrowAsException(const ara::core::ErrorCode &errorCode)	
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:	noexcept(false)	
Header file:	#include "ara/com/com_error_domain.h"	
Description:	Creates a new instance of ComException from errorCode and throws it as a C++ exception.	

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

[SWS_CM_11334]{DRAFT} ⌈

Kind:	function
Symbol:	GetComErrorDomain()

▽

△

Scope:	namespace ara::com	
Syntax:	`constexpr ara::core::ErrorDomain& GetComErrorDomain () noexcept;`	
Return value:	ara::core::ErrorDomain &	Return a reference to the global ComErrorDomain object.
Exception Safety:	noexcept	
Header file:	#include "ara/com/com_error_domain.h"	
Description:	Returns a reference to the global ComErrorDomain object.	

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

[SWS_CM_11335]{DRAFT} ⌈

Kind:	function	
Symbol:	MakeErrorCode(ara::com::ComErrc code, ara::core::ErrorDomain::SupportDataType data)	
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 ErrorCode object.
Exception Safety:	noexcept	
Header file:	#include "ara/com/com_error_domain.h"	
Description:	Creates an instance of ErrorCode.	

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

[SWS_CM_11266]{DRAFT} **Definition of Application Errors** ⌈Each `ApApplicationError` references an `ApApplicationErrorDomain`. The error domain corresponding `ApApplicationErrorDomain` shall be defined as specified in [16]. The corresponding enumeration shall contain an entry for each `ApApplicationError` referencing this `ApApplicationErrorDomain` using the `shortName` of the `ApApplicationError` as symbol and the `errorCode` of the `ApApplicationError` as value:

```
1
2 enum class <ApApplicationErrorDomain.SN>Errc : ara::core::ErrorDomain::
      CodeType
3 {
4         <ApApplicationError.SN> = <ApApplicationError.errorCode>,
5
6 };
```

⌋(*RS_CM_00211*, *RS_AP_00114*, *RS_AP_00119*, *RS_AP_00127*)

[SWS_CM_11268] Definition general ara::com::raw errors ⌈General ara::com::raw errors shall be defined in the error domain `ara::com::raw::RawErrorDomain` in accordance with [16].

⌋(*RS_AP_00130*)

[SWS_CM_99025]{DRAFT} **Raw errors domain** ⌈Error domain to describe ara::com errors related to the RawDataStreamInterface `ara::com::raw::RawErrorDomain` shall be defined. It shall have the shortname Raw and the identifier 0x8000'0000'0000'1280.⌋*(RS_AP_00130)*

[SWS_CM_12367] ⌈

Kind:	enumeration	
Symbol:	RawErrc	
Scope:	namespace ara::com::raw	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class RawErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kStreamNotConnected= 1	Trying to use a raw data stream without an established connection.
	kCommunicationTimeout= 2	The operation was not successful and timed out.
	kConnectionRefused= 3	The target address was not listening for connections or refused the connection request.
	kAddressNotAvailable= 4	The specified address is not available from the local machine.
	kStreamAlreadyConnected= 5	The specified connection is already connected.
	kConnectionClosedByPeer= 6	Network error. The established connection has been shut down during writing (POSIX EPIPE).
	kPeerUnreachable= 7	Network error. The peer is unreachable (POSIX ENETUNREACH).
	kConnectionAborted= 8	Network error. The incoming connection was aborted (POSIX ECONNABORTED).
	kInterruptedBySignal= 9	System error. Operation interrupted by system (POSIX EINTR).
	kConnectionCreationFailed= 10	Permission to create a connection is denied. (POSIX EACCES)
Header file:	#include "ara/com/raw/raw_error_domain.h"	
Description:	The RawErrc enumeration defines the error codes for the RawErrorDomain. .	

⌋*(RS_AP_00130, RS_AP_00122, RS_AP_00127)*

[SWS_CM_11291]{DRAFT} ⌈

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

⌋*(RS_AP_00130, RS_AP_00122, RS_AP_00127)*

[SWS_CM_11292]{DRAFT} ⌈

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

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

[SWS_CM_11298]{DRAFT} ⌈

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

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

[SWS_CM_11299]{DRAFT} ⌈

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

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

[SWS_CM_11293]{DRAFT} ⌈

Kind:	class
Symbol:	RawErrorDomain
Scope:	namespace ara::com::raw

▽

△

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

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

[SWS_CM_11295]{DRAFT} ⌈

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

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

[SWS_CM_11296]{DRAFT} ⌈

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

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

[SWS_CM_11297]{DRAFT} ⌈

Kind:	function	
Symbol:	ThrowAsException(const ara::core::ErrorCode &errorCode)	
Scope:	class ara::com::raw::RawErrorDomain	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const noexcept(false) override;`	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	noexcept(false)	

▽

△

Header file:	#include "ara/com/raw/raw_error_domain.h"
Description:	Creates a new instance of RawException from errorCode and throws it as a C++ exception.

⌋*(RS_AP_00120, RS_AP_00121, RS_AP_00130)*

[SWS_CM_99026]{DRAFT} **E2E errors domain** ⌈Error domain to describe E2E related ara::com errors `ara::com::e2e::E2EErrorDomain` shall be defined. It shall have the shortname E2E and the identifier 0x8000'0000'0000'1268.⌋*(RS_AP_00130)*

[SWS_CM_10474]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	E2EErrc	
Scope:	namespace ara::com::e2e	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class E2EErrc : 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.
	kNotAvailable= 4	No value has been received yet (e.g. during initialization). This is used as the initialization value for the buffer, it is not returned by any E2E profile.
	kNoNewData= 5	No new data is available.
Header file:	#include "ara/com/e2e/e2e_error_domain.h"	
Description:	The E2EErrc enumeration defines the error codes for the E2EErrorDomain. .	

⌋*(RS_AP_00130)*

[SWS_CM_99023]{DRAFT} **Definition general ara::com::cg errors** ⌈General ara::com::cg errors shall be defined in the error domain `ara::com::cg::CgErrorDomain` in accordance with [16].⌋*(RS_AP_00130)*

[SWS_CM_99024]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	CgErrc	
Scope:	namespace ara::com::cg	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class CgErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kCommunicationGroupNotActive= 1	Commincation Group not active/connected by a Server.
	kNoClients= 2	No communication group clients.
	kWrongClientAddress= 3	Wrong client address.
	kBindingError= 4	Error at technology binding.

▽

△

	kMemoryError= 5	Memory Error.
	kServerExists= 6	Other server already connected to communication group.
Header file:	#include "ara/com/cg/cg_error_domain.h"	
Description:	The CgErrc enumeration defines the error codes for the CgErrorDomain. .	

⌋(*RS_AP_00130*)

[SWS_CM_99027]{DRAFT} **Cg errors domain** ⌈Error domain to describe ara::com errors related to the Communication Groups `ara::com::cg::CgErrorDomain` shall be defined. It shall have the shortname Cg and the identifier 0x8000'0000'0000'1270.⌋ (*RS_AP_00130*)

8.1.2.7 E2E Related Data Types

Some data types are used only in context of e2e-protected communication of `event`s.

[SWS_CM_90421]{DRAFT} **ara::com::e2e::ProfileCheckStatus** ⌈The Communication Management shall provide an enumeration `ara::com::e2e::ProfileCheckStatus` which represents the results of the check of a single `sample`:

- `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 GetProfileCheckStatus if `EndToEndTransformationComSpecProps.disableEndToEndCheck` is set to TRUE

```
enum class ProfileCheckStatus : std::uint8_t
{
    kOk,
    kRepeated,
    kWrongSequence,
    kError,
    kCheckDisabled
};
```

Results of E2E are described in [PRS_E2E_00322] and [PRS_E2E_00677] of [4].⌋ (*RS_E2E_08534*, *RS_AP_00114*, *RS_AP_00115*, *RS_AP_00119*)

[SWS_CM_90426]{DRAFT} **Mapping of `ProfileCheckStatus`** ⌈The E2E profile independent results according to [PRS_E2E_00677] shall be mapped to the enumeration literals of `ara::com::e2e::ProfileCheckStatus` as described in Table 8.1.⌋ *(RS_E2E_08534, RS_AP_00114, RS_AP_00115, RS_AP_00119)*

Enumeration literal of `ProfileCheckStatus`	Profile independent result of `E2E_Check()`
`kOk`	OK
`kRepeated`	REPEATED
`kWrongSequence`	WRONGSEQUENCE
`kError`	WRONGCRC
n/a	NONEWDATA
`kCheckDisabled`	n/a

Table 8.1: Mapping of `ProfileCheckStatus`

The E2E state machine `SMState` is determined by checking a history of `ProfileCheckStatus`es. The current value of `SMState` mirrors the current state of the E2E supervision, but is not necessarily applicable to all samples received during the last update.

[SWS_CM_90422]{DRAFT} **ara::com::e2e::SMState** ⌈The Communication Management shall provide an enumeration `ara::com:e2e::SMState` which represents in what state is the E2E supervision after the most recent check of the `sample`(s) of a received `sample` of the `event`. If `SMState` is `Valid`, and the `GetProfileCheckStatus` did not result in Error then the last checked `sample` can be used.

- `kValid`: Communication of the `sample`s of this `event` 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 few data where the E2E check yielded OK or to many 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 `GetE2EStateMachineState` if `EndToEndTransformationComSpecProps.disableEndToEndStateMachine` is set to TRUE.

```
enum class SMState : std::uint8_t
{
    kValid,
    kNoData,
    kInit,
    kInvalid,
    kStateMDisabled
};
```

Results of E2E state machine are described in [PRS_E2E_00322] and [PRS_E2E_-00678] of [4].⌋*(RS_E2E_08534, RS_AP_00114, RS_AP_00115, RS_AP_00119)*

[SWS_CM_90427]{DRAFT} **Mapping of `SMState`** ⌈The communication channel status according to [PRS_E2E_00678] shall be mapped to the enumeration literals of `SM-State` as described in Table 8.2.⌋*(RS_E2E_08534, RS_AP_00114, RS_AP_00115, RS_AP_00119)*

Enumeration literal of `SMState`	communication channel status
`kValid`	VALID
`kNoData`	NODATA, DEINIT
`kInit`	INIT
`kInvalid`	INVALID
`kStateMDisabled`	n/a

Table 8.2: Mapping of `SMState`

8.1.2.8 Raw Data Stream Data Type

[SWS_CM_11300]{DRAFT} ⌈

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

⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_CM_11301]{DRAFT} ⌈

Kind:	variable
Symbol:	data
Scope:	struct ara::com::raw::ReadDataResult
Type:	ara::com::SamplePtr< std::uint8_t >
Syntax:	`ara::com::SamplePtr<std::uint8_t> data;`
Header file:	#include "ara/com/raw/raw_data_stream.h"
Description:	Pointer to the read data (SamplePtr to get std::unique_ptr semantics).

⌋*(RS_CM_00410, RS_CM_00411, RS_CM_00412)*

[SWS_CM_11302]{DRAFT} ⌈

Kind:	variable
Symbol:	numberOfBytes

▽

△

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

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

8.1.3 API Reference

The ServiceInterface description is the input for the generation of the service API header files content.

The proxy and skeleton header files contain different classes representing the ServiceInterface itself and its elements event, method and field.

[SWS_CM_00002]{DRAFT} **Service skeleton class** ⌈The Communication Management shall provide the definition of a C++ class named <name>Skeleton in the service skeleton header file within the namespace defined by [SWS_CM_01006], where <name> is the ServiceInterface.shortName in upper camel case format.

```
class UpperCamelCase(<ServiceInterface.shortName>)Skeleton {
...
};
```

⌋(*RS_CM_00101*, *RS_AP_00114*, *RS_AP_00122*)

[SWS_CM_00003]{DRAFT} **Service skeleton Event class** ⌈For each VariableDataPrototype defined in the ServiceInterface in the role event the definition of a C++ class using the shortName in upper camel case format of the VariableDataPrototype shall be provided in the service skeleton header file within the namespace defined by [SWS_CM_01009].

```
class UpperCamelCase(<VariableDataPrototype.shortName>) {
...
};
```

⌋(*RS_CM_00201*, *RS_AP_00114*)

[SWS_CM_00007]{DRAFT} **Service skeleton Field class** ⌈For each Field defined in the ServiceInterface in the role field the definition of a C++ class using the shortName in upper camel case format of the Field shall be provided in the service skeleton header file within the namespace defined by [SWS_CM_01031].

```
class UpperCamelCase(<Field.shortName>) {
...
};
```

⌋(*RS_CM_00219*, *RS_AP_00114*)

[SWS_CM_00004]{DRAFT} **Service proxy class** ⌈The Communication Management shall provide the definition of a C++ class named <name>Proxy in the service proxy header file within the namespace defined by [SWS_CM_01007], where <name> is the ServiceInterface.shortName in upper camel case format.

```
class UpperCamelCase(<ServiceInterface.shortName>)Proxy {
...
};
```

⌋(*RS_CM_00102*, *RS_AP_00114*, *RS_AP_00122*)

[SWS_CM_00005]{DRAFT} **Service proxy Event class** ⌈For each VariableDataPrototype defined in the ServiceInterface in the role event the definition of

a C++ class using the shortName in upper camel case format of the VariableDataPrototype shall be provided in the service proxy header file within the namespace defined by [SWS_CM_01009].

```
class UpperCamelCase(<VariableDataPrototype.shortName>) {
...
};
```

⌋(*RS_CM_00103*, *RS_AP_00114*)

[SWS_CM_00006]{DRAFT} **Service proxy Method class** ⌈For each ClientServerOperation defined in the ServiceInterface in the role method the definition of a C++ class using the shortName in upper camel case format of the ClientServerOperation shall be provided in the service proxy header file within the namespace defined by [SWS_CM_01015].

```
class UpperCamelCase(<ClientServerOperation.shortName> {
...
};
```

⌋(*RS_CM_00212*, *RS_CM_00213*, *RS_AP_00114*)

[SWS_CM_00008]{DRAFT} **Service proxy Field class** ⌈For each Field defined in the ServiceInterface in the role field the definition of a C++ class using the shortName in upper camel case format of the ServiceInterface shall be provided in the service proxy header file within the namespace defined by [SWS_CM_01031].

```
class UpperCamelCase(<Field.shortName>) {
...
};
```

⌋(*RS_CM_00216*, *RS_AP_00114*)

[SWS_CM_99028]{DRAFT} **Types of APIs - Communication and Service Discovery APIs** ⌈There are two categories of APIs: Service Discovery API and Communication API.

Service Discovery API : These APIs are used in service discovery process. The following APIs are Service Discovery APIs:

- OfferService()
- StopOfferService()
- RegisterGetHandler()
- RegisterSetHandler()
- FindService()
- StartFindService()
- StopFindService()
- GetHandle()

- `Subscribe()`
- `Unsubscribe()`
- `GetSubscriptionState()`
- `SetSubscriptionStateChangeHandler()`
- `UnsetSubscriptionStateChangeHandler()`
- `SetReceiveHandler()`
- `UnsetReceiveHandler()`

Communication API : These APIs are used in communication between Client and Server. The following APIs are Communication APIs:

- `Send()`
- `Allocate()`
- `Update()`
- `GetNewSamples()`
- `GetFreeSampleCount()`
- `Method call operator()`
- `Get()`
- `Set()`

⌋*()*

[SWS_CM_00009]{DRAFT} **Re-entrancy and thread-safety - General** ⌈The concurrent invocation of communication APIs shall be allowed irrespective of the class instance. - I.e., concurrent invocation of *different* member functions shall be allowed for the same class instance and for *different* class instances.

The concurrent invocation of service discovery APIs shall be allowed for *different* class instances and shall not be allowed for same class instances.

Only communication APIs shall be thread-safe against each other (for same class instance).

Service Discovery APIs shall be NOT thread-safe against each other, or against Communication APIs (for same proxy/skeleton instance).⌋*()*

The following sub-chapters describe the content of the previously defined classes.

8.1.3.1 Object Creation via Named Constructor Approach

The Named Constructor approach enables exception-less error reporting for object construction. Since service skeletons and service proxies can be created using a `Named Constructor`, this section describes the general requirements of this approach. For the service skeleton and service proxy creation C++ API reference, see chapter 8.1.3.3 and 8.1.3.11, respectively.

[SWS_CM_11326]{DRAFT} **Creation of an object using Named Constructor approach** ⌈The ClassToBeCreated shall provide a static member function Create() returning the constructed object embedded in an `ara::core::Result`. This function first performs all operations for constructing an object of ClassToBeCreated, which may fail or result in an error. E.g. parameter checks or resource allocation may fail. If an error occurs during these operations, the error is returned as an ara::core::ErrorCode in the ara::core::Result. If no error occurs, the created object is returned as a value in the ara::core::Result. The value object can then be considered as valid. The function shall not throw an exception.

```
static ara::core::Result<ClassToBeCreated>
        Create(/* construction  arguments */)
        noexcept;
```

Unless a potentially-throwing constructor shall be available for ClassToBeCreated, only the Create() function shall be public for the user. All regular constructors shall be private.⌋*(RS_CM_00101, RS_AP_00114, RS_AP_00139, RS_AP_00128, RS_AP_00132, RS_AP_00127, RS_AP_00139)*

8.1.3.2 Offer service

For the functional description of the service offering API, see chapter 7.10.1.

[SWS_CM_00101] Method to offer a service ⌈The Communication Management shall provide an `OfferService` method as part of the `ServiceSkeleton` class to offer a service to applications.

```
ara::core::Result<void> OfferService();
```

If the offered service contains a `Field`, and the field value is not valid according to [SWS_CM_00128] when `OfferService()` is called, the service shall not be offered, and the error code `ComErrc::kFieldValueIsNotValid` shall be returned in the `Result` type.

If the offered service contains a `Field` that is defined with `hasSetter=true`, and no `SetHandler` has been registered yet, the service shall not be offered, and the error code `ComErrc::kSetHandlerNotSet` shall be returned in the `Result` type. See [SWS_CM_00129]⌋*(RS_CM_00101, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00010]{DRAFT} **Re-entrancy and thread-safety - `OfferService`** ⌈`OfferService` shall be re-entrant and thread-safe for *different* `ServiceSkeleton`

class instances. When called re-entrant or concurrently on the same `ServiceSkeleton` class instance, the behavior is undefined.⌋*(RS_CM_00101, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00111] Method to stop offering a service ⌈The Communication Management shall provide a `StopOfferService` method as part of the `ServiceSkeleton` class to stop offering services to applications.

```
void StopOfferService();
```

⌋*(RS_CM_00105, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00011]{DRAFT} **Re-entrancy and thread-safety- `StopOfferService`** ⌈`StopOfferService` shall be re-entrant and thread-safe for *different* `ServiceSkeleton` class instances. When called re-entrant or concurrently on the same `ServiceSkeleton` class instance, the behavior is undefined.⌋*(RS_CM_00105, RS_AP_00114, RS_AP_00120)*

8.1.3.3 Service skeleton creation

For the functional description of the service skeleton creation API, see chapter 7.10.2.

[SWS_CM_00130] Creation of service skeleton using Instance ID ⌈The Communication Management shall provide a constructor for each specific `ServiceSkeleton` class taking two arguments:

- `InstanceIdentifier`: The identifier of a specific instance of a service, needed to distinguish different instances of exactly the same service in the system. See [SWS_CM_00302] for the type definition.
- `MethodCallProcessingMode`: As a default argument, this is the mode of the service implementation for processing service method invocations with `kEvent` as default value. See [SWS_CM_00301] for the type definition and [SWS_CM_00198] for more details on the behavior.

```
ServiceSkeleton(
  ara::com::InstanceIdentifier instanceID,
  ara::com::MethodCallProcessingMode mode =
          ara::com::MethodCallProcessingMode::kEvent
);
```

⌋*(RS_CM_00101, RS_AP_00114, RS_AP_00121, RS_AP_00145)*

[SWS_CM_10435] Exception-less creation of service skeleton using Instance ID ⌈The Communication Management shall provide a non-throwing constructor for each specific `ServiceSkeleton` class using the Named Constructor idiom according to [SWS_CM_11326]. The Named Constructor shall be called Create() and shall take two arguments:

- `InstanceIdentifier`: The identifier of a specific instance of a service, needed to distinguish different instances of exactly the same service in the system. See [SWS_CM_00302] for the type definition.
- `MethodCallProcessingMode`: As a default argument, this is the mode of the service implementation for processing service method invocations with `kEvent` as default value. See [SWS_CM_00301] for the type definition and [SWS_CM_00198] for more details on the behavior.

```
static ara::core::Result<ServiceSkeleton> Create(
           const ara::com::InstanceIdentifier &instanceID,
         ara::com::MethodCallProcessingMode mode =
       ara::com::MethodCallProcessingMode::kEvent) noexcept ;
```

In case e2e-protected methods are used by the service, and a MethodCallProcessingMode of kEvent is passed to the constructor, an error code kWrongMethodCallProcessingMode shall be returned in the Result. See [SWS_CM_10467]

In case of a `Grant` enforcement failure, an error code `ComErrc::kGrantEnforcementError` shall be returned in the Result. See [SWS_CM_90005].

⌋(*RS_CM_00101, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00139, RS_AP_00128, RS_AP_00132, RS_AP_00127, RS_AP_00139, RS_AP_00145*)

[SWS_CM_00152] Creation of service skeleton using Instance Spec ⌈The Communication Management shall provide a constructor for each specific `ServiceSkeleton` class taking two arguments:

- `InstanceSpecifier`: The specifiers of a specific instance of a service, needed to distinguish different instances of exactly the same service in the system. See [SWS_CORE_08001] for the type definition.
- `MethodCallProcessingMode`: As a default argument, this is the mode of the service implementation for processing service method invocations with `kEvent` as default value. See [SWS_CM_00301] for the type definition and [SWS_CM_00198] for more details on the behavior.

```
ServiceSkeleton(
  ara::core::InstanceSpecifier instanceSpec,
  ara::com::MethodCallProcessingMode mode =
          ara::com::MethodCallProcessingMode::kEvent
);
```

⌋(*RS_CM_00101, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00127, RS_AP_00137, RS_AP_00145*)

[SWS_CM_10436] Exception-less creation of service skeleton using Instance Spec ⌈The Communication Management shall provide a non-throwing constructor for each specific `ServiceSkeleton` class using the Named Constructor idiom according to [SWS_CM_11326]. The Named Constructor shall be called Create() and shall take two arguments:

- `InstanceSpecifier`: The specifiers of a specific instance of a service, needed to distinguish different instances of exactly the same service in the system. See [SWS_CORE_08001] for the type definition.
- `MethodCallProcessingMode`: As a default argument, this is the mode of the service implementation for processing service method invocations with `kEvent` as default value. See [SWS_CM_00301] for the type definition and [SWS_CM_00198] for more details on the behavior.

```
static ara::core::Result<ServiceSkeleton> Create(
           const ara::core::InstanceSpecifier &instanceSpec,
          ara::com::MethodCallProcessingMode mode =
       ara::com::MethodCallProcessingMode::kEvent) noexcept;
```

In case e2e-protected methods are used by the service, and a MethodCallProcessingMode of kEvent is passed to the constructor, an error code kWrongMethodCallProcessingMode shall be returned in the Result. See [SWS_CM_10467].

In case of a `Grant` enforcement failure, an error code `ComErrc::kGrantEnforcementError` shall be returned in the Result. See [SWS_CM_90005].

⌋(*RS_CM_00101, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00139, RS_AP_00128, RS_AP_00132, RS_AP_00127, RS_AP_00137, RS_AP_00139, RS_AP_00145*)

[SWS_CM_00153] Creation of service skeleton using Instance ID Container ⌈The Communication Management shall provide a constructor for each specific `ServiceSkeleton` class taking two arguments:

- `InstanceIdentifierContainer`: The container of instances of a service, each instance element needed to distinguish different instances of exactly the same service in the system. See [SWS_CM_00319] for the type definition.
- `MethodCallProcessingMode`: As a default argument, this is the mode of the service implementation for processing service method invocations with `kEvent` as default value. See [SWS_CM_00301] for the type definition and [SWS_CM_00198] for more details on the behavior.

```
ServiceSkeleton(
  ara::com::InstanceIdentifierContainer instanceIDs,
  ara::com::MethodCallProcessingMode mode =
          ara::com::MethodCallProcessingMode::kEvent
);
```

⌋(*RS_CM_00101, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00145*)

[SWS_CM_10437] Exception-less creation of service skeleton using Instance ID Container ⌈The Communication Management shall provide a non-throwing constructor for each specific `ServiceSkeleton` class using the Named Constructor idiom according to [SWS_CM_11326]. The Named Constructor shall be called Create() and shall take two arguments:

- `InstanceIdentifierContainer`: The container of instances of a service, each instance element needed to distinguish different instances of exactly the same service in the system. See [SWS_CM_00319] for the type definition.
- `MethodCallProcessingMode`: As a default argument, this is the mode of the service implementation for processing service method invocations with `kEvent` as default value. See [SWS_CM_00301] for the type definition and [SWS_CM_00198] for more details on the behavior.

```
static ara::core::Result<ServiceSkeleton> Create(
           const ara::com::InstanceIdentifierContainer &instanceIDs,
          ara::com::MethodCallProcessingMode mode =
       ara::com::MethodCallProcessingMode::kEvent) noexcept;
```

In case e2e-protected methods are used by the service, and a MethodCallProcessingMode of kEvent is passed to the constructor, an error code kWrongMethodCallProcessingMode shall be returned in the Result. See [SWS_CM_10467].

In case of a `Grant` enforcement failure, an error code `ComErrc::kGrantEnforcementError` shall be returned in the Result. See [SWS_CM_90005].

⌋(*RS_CM_00101*, *RS_AP_00114*, *RS_AP_00115*, *RS_AP_00121*, *RS_AP_00139*, *RS_AP_00128*, *RS_AP_00132*, *RS_AP_00127*, *RS_AP_00139*, *RS_AP_00145*)

[SWS_CM_00134] Copy semantics of service skeleton class ⌈The Communication Management shall disable the generation of the copy constructor and the copy assignment operator for each specific `ServiceSkeleton` class.

```
ServiceSkeleton(const ServiceSkeleton&) = delete;
ServiceSkeleton& operator=(const ServiceSkeleton&) = delete;
```

⌋(*RS_CM_00101*, *RS_AP_00114*, *RS_AP_00145*, *RS_AP_00147*)

[SWS_CM_00135] Move semantics of service skeleton class ⌈The Communication Management shall provide the possibility to move construct and move assign a `ServiceSkeleton` instance from another instance.

```
ServiceSkeleton(ServiceSkeleton &&);
ServiceSkeleton& operator=(ServiceSkeleton &&);
```

⌋(*RS_CM_00101*, *RS_AP_00114*, *RS_AP_00145*, *RS_AP_00147*)

[SWS_CM_11370]{DRAFT} **ServiceSkeleton destructor** ⌈The Communication Management shall provide a destructor for the `ServiceSkeleton`.

```
~ServiceSkeleton();
```

⌋(*RS_AP_00114*, *RS_AP_00145*)

8.1.3.4 Send event

Inside the specific `Event` class belonging to the specific `ServiceSkeleton` class a `Send` method shall be provided to initiate sending the corresponding event .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` method shall be provided in two ways: One where the application is owner of the data and the `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_00162] Send event where application is responsible for the data ⌈The `Send` method of the specific `Event` class where the application is responsible for the data and the Communication Management creates a copy for sending takes in the input parameter `data`, the data to send and sends it to all subscribed applications. This version of the `Send` method shall be used whenever the application wants to work further with the data.

```
ara::core::Result<void> Event::Send(const SampleType &data);
```

If not successful, `Send()` shall return an `ara::core::ErrorCode` from the `ara::com::ComErrorDomain` indicating the error. The following errors are possible:

- `ComErrc::kServiceNotOffered`: Service not offered.
- `ComErrc::kCommunicationLinkError`: Communication link is broken.
- `ComErrc::kCommunicationStackError`: Communication Stack Error, e.g. network stack, network binding, or communication framework reports an error.

⌋(*RS_CM_00201*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00121*)

[SWS_CM_90437] Send event where Communication Management is responsible for the data ⌈The `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 takes in the input parameter `data`, the data to send and sends it to all subscribed applications.

```
ara::core::Result<void> Event::Send(ara::com::SampleAllocateePtr
                                    <SampleType> data);
```

Before sending the event the corresponding data has to be requested from the Communication Management (see [SWS_CM_90438]) and filled with the respective data.

This version of the `Send` method shall be used whenever the data is created explicitly for sending and no further processing is happening afterward by the application itself.

If not successful, `Send()` shall return an `ara::core::ErrorCode` from the `ara::com::ComErrorDomain` indicating the error. The following errors are possible:

- `ComErrc::kServiceNotOffered`: Service not offered.
- `ComErrc::kCommunicationLinkError`: Communication link is broken.

- `ComErrc::kCommunicationStackError`: Communication Stack Error, e.g. network stack, network binding, or communication framework reports an error.

⌋(*RS_CM_00201*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00121*)

[SWS_CM_00012]{DRAFT} **Re-entrancy and thread-safety - `Send`** ⌈`Send` shall be re-entrant and thread-safe for *different* `Event` class instances. When called re-entrant or concurrently on the same Event class instance, the behavior is undefined.⌋(*RS_CM_00201*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00121*)

[SWS_CM_90438] Allocating data for event transfer ⌈Data shall be requested by calling the `Allocate` method of the specific `Event` class. By calling the `Send` method with the data, it is ensured that the data will be freed by the Communication Management.

There are two error codes that shall be returned in the `Result` of `Allocate()`:

- `ComErrc::kSampleAllocationFailure`: If the allocation of the shared memory fails (i.e., failure to retrieve/allocate a shared slot for a sample).
- `ComErrc::kIllegalUseOfAllocate`: If the allocation is done via custom allocator (i.e., not via shared memory allocation).The error shall be logged.

```
ara::core::Result<ara::com::SampleAllocateePtr<SampleType>>
   Event::Allocate();
```

⌋(*RS_CM_00201*, *RS_AP_00114*, *RS_AP_00115*, *RS_AP_00120*)

See [SWS_CM_00308] for the type definition of `SampleAllocateePtr` and ARA-ComAPI explanatory document [1] for more details on the behavior.

Since the `SampleAllocateePtr` pointer type behaves like a `std::unique_ptr`, the ownership of the pointer has to be transferred via `std::move` for utilizing zero-copy optimizations.

[SWS_CM_00013]{DRAFT} **Re-entrancy and thread-safety - `Allocate`** ⌈`Allocate` shall be re-entrant and thread-safe for *different* `Event` class instances. When called re-entrant or concurrently on the same Event class instance, the behavior is undefined.⌋(*RS_CM_00201*, *RS_AP_00114*, *RS_AP_00115*, *RS_AP_00120*)

8.1.3.5 Send Trigger

Inside the specific `Trigger` class belonging to the specific `ServiceSkeleton` class a `Send` method shall be provided to initiate sending the corresponding trigger.

[SWS_CM_00721]{DRAFT} **Send trigger** ⌈The `Send` method of the specific `Trigger` class send trigger to all subscribed applications.

```
ara::core::Result<void> Trigger::Send();
```

If not successful, `Send()` shall return an `ara::core::ErrorCode` from the `ara::com::ComErrorDomain` indicating the error. The following errors are possible:

- `ComErrc::kServiceNotOffered`: Service not offered.
- `ComErrc::kCommunicationLinkError`: Communication link is broken.
- `ComErrc::kCommunicationStackError`: Communication Stack Error, e.g. network stack, network binding, or communication framework reports an error.

⌋(*RS_CM_00201*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00121*)

[SWS_CM_00722]{DRAFT} **Re-entrancy and thread-safety - `Send`** ⌈`Send` shall be re-entrant and thread-safe for different `Trigger` class instances. When called re-entrant or concurrently on the same `Trigger` class instance, the behavior is undefined.⌋(*RS_CM_00201*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00121*)

8.1.3.6 Provide a service method

[SWS_CM_00191]{DRAFT} **Provision of method** ⌈A pure virtual method shall be defined inside the specific `ServiceSkeleton` class for each provided method of the service.
The name of this method and its parameters are derived from the signature of the provided service method.
The service method input parameters shall become input parameters of the respective method defined inside the `ServiceSkeleton` class.
An `Output` type combining the possible output parameters shall be provided inside the `ServiceSkeleton` class.
The method shall return an `ara::core::Future` object wrapping the output parameters as result.
A corresponding subclass providing implementations for the methods shall be created to implement the methods of a respective `ServiceSkeleton`.

```
struct Method1Output {
  TypeOutputParameter1 output1;
  TypeOutputParameter2 output2;
  ...
};

virtual ara::core::Future <Method1Output> Method1(
  TypeInputParameter1 input1,
  TypeInputParameter2 input2,
  ...
) = 0;
```

⌋(*RS_CM_00211*, *RS_AP_00114*, *RS_AP_00138*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00138*)

[SWS_CM_90434]{DRAFT} **Provision of a `Fire and Forget method`** ⌈A pure virtual method shall be defined inside the specific `ServiceSkeleton` class for each provided `Fire and Forget method` of the service.
The name of this method and its parameters are derived from the signature of the provided `Fire and Forget method`.
The `Fire and Forget method` input parameters shall become input parameters of the respective method defined inside the `ServiceSkeleton` class.
The `Fire and Forget method` shall have no return values.
A corresponding subclass providing implementations for the `Fire and Forget method`s shall be created to implement the `Fire and Forget method` of a respective `ServiceSkeleton`.

```
virtual void FireForgetMethod1(
  TypeInputParameter1 input1,
  TypeInputParameter2 input2,
  ...
) = 0;
```

⌋*(RS_CM_00225, RS_AP_00114)*

[SWS_CM_00017]{DRAFT} **Re-entrancy and thread-safety - `ServiceSkeleton` method implementation** ⌈The `ServiceSkeleton` method implementation shall be re-entrant and thread-safe in case the `ServiceSkeleton` instance has been created in event-driven concurrent mode(`kEvent`). - The `ServiceSkeleton` method implementation may be non-re-entrant and non-thread-safe otherwise (i.e., in event-driven sequential mode (`kEventSingleThread`) and in polling mode (`kPoll`)).⌋ *(RS_CM_00211, RS_AP_00114, RS_AP_00138, RS_AP_00128, RS_AP_00127, RS_AP_00138)*

8.1.3.7 Processing of service methods

For the functional description of the processing of service methods API, see chapter 7.10.3.

[SWS_CM_00198]{DRAFT} **Set service method processing mode** ⌈With the instantiation of a specific `ServiceSkeleton` class, the mode for processing service method invocations is set by providing an `ara::com::MethodCallProcessingMode` as a parameter of the constructor. The mode 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 data type representing the processing modes is defined by [SWS_CM_00301].
The following processing modes shall be supported:

- **Polling** (enumeration element `kPoll`)
- **Event-driven, concurrent** (enumeration element `kEvent`)
- **Event-driven, sequential** (enumeration element `kEventSingleThread`)

⌋(*RS_CM_00211*, *RS_AP_00114*, *RS_AP_00115*, *RS_AP_00120*)

[SWS_CM_00199]{DRAFT} **Process Service method invocation** ⌈Inside the specific `ServiceSkeleton` class, a `ProcessNextMethodCall` method shall be provided. This method 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`.
The method shall return an `ara::core::Future` object wrapping a `bool` parameter as return value. A returned value `true` indicates that there is at least one pending invocation, returning `false` indicates the opposite. The returned `ara::core::Future` becomes ready, after the service method (see [SWS_CM_00191] or [SWS_CM_90434]) invoked due to `ProcessNextMethodCall()` has completed.

```
ara::core::Future<bool> ProcessNextMethodCall();
```

⌋(*RS_CM_00211*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00138*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00138*)

[SWS_CM_11350]{DRAFT} **Execution Context for process service method invocation** ⌈For the `ProcessNextMethodCall` method described in [SWS_CM_00199] a second overload with an additional input parameter. This parameter shall provide an executioner object in which any asynchronous computation spawn by `ProcessNextMethodCall` shall be invoked. The minimum behavior of the Execution Context is defined in [SWS_CM_11364].For the first overload without an execution context argument an implementation defined default execution context (like in previous AUTOSAR releases) shall be used.

```
template<typename ExecutorT>
ara::core::Future<bool> ProcessNextMethodCall(ExecutorT &&executor);
```

⌋(*RS_CM_00211*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00138*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00138*)

[SWS_CM_11351]{DRAFT} **Error behaviour of provided Execution Context for process service method invocation** ⌈In case a `ProcessNextMethodCall()` cannot be executed with the provided executor (e.g. because of resource problem) an `ComErrc::kCouldNotExecute` error shall be raised in all cases.⌋(*RS_CM_00211*, *RS_CM_00212*, *RS_CM_00213*, *RS_CM_00214*, *RS_AP_00114*, *RS_AP_00119*, *RS_AP_00127*)

[SWS_CM_10362]{DRAFT} **Raising checked errors for application errors** ⌈Whenever on the skeleton side of a service method an `ApApplicationError` – according to the interface description in the Manifest – is detected, the corresponding `ara::core::ErrorCode` representing this `ApApplicationError` (see [SWS_CM_11266]) shall be stored into the `ara::core::Promise` object, from which the `ara::core::Future` is returned to the caller.⌋(*RS_CM_00211*, *RS_CM_00212*, *RS_CM_00213*, *RS_CM_00214*, *RS_AP_00114*, *RS_AP_00119*, *RS_AP_00127*)

8.1.3.8 Registering get handlers for fields

For the functional description of the registering get handlers for fields API, see chapter 7.10.4.

[SWS_CM_00114]{DRAFT} **Registering Getters** ⌈Inside the specific `Field` class belonging to the specific `ServiceSkeleton` class a `RegisterGetHandler` method shall be provided to give the possibility to register a `GetHandler`.

```
ara::core::Result<void> RegisterGetHandler(
        std::function<ara::core::Future<FieldType>(
                )> getHandler);
```

⌋*(RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127)*

[SWS_CM_11360]{DRAFT} **Execution Context for registering Getters** ⌈For the `RegisterGetHandler` method described in [SWS_CM_00114] a second overload with an additional input parameter shall be provided. This parameter shall provide an executioner object in which any asynchronous computation spawn by `RegisterGetHandler` shall be invoked. The minimum behavior of the Execution Context is defined in [SWS_CM_11364].

```
template<typename ExecutorT>
ara::core::Result<void> RegisterGetHandler(
  std::function<ara::core::Future<FieldType>(
          )> getHandler, ExecutorT&& executor);
```

For the first overload without an execution context argument an implementation defined default execution context (like in previous AUTOSAR releases) shall be used.⌋ *(RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127, RS_AP_00138)*

[SWS_CM_11361]{DRAFT} **Error behaviour of provided Execution Context for registering Getters** ⌈In case a `RegisterGetHandler()` cannot be executed with the provided executor (e.g. because of resource problem) a `ComErrc::kCouldNotExecute` error shall be raised in all cases.⌋*(RS_CM_00211, RS_CM_00212, RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127)*

[SWS_CM_00115]{DRAFT} **Existence of RegisterGetHandler method** ⌈The existence of `RegisterGetHandler` as part of the `Field` class shall be controlled by `Field.hasGetter`.⌋*(RS_CM_00218, RS_AP_00114)*

[SWS_CM_00014]{DRAFT} **Re-entrancy and thread-safety - `RegisterGetHandler`** ⌈`RegisterGetHandler` shall be re-entrant and thread-safe for *different* `Field` class instances. When called re-entrant or concurrently on the same `Field` class instance, the behavior is undefined.⌋*(RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127)*

8.1.3.9 Registering set handlers for fields

For the functional description of the registering set handlers for fields API, see chapter 7.10.5.

[SWS_CM_00116]{DRAFT} **Registering Setters** ⌈Inside the specific `Field` class belonging to the specific `ServiceSkeleton` class a `RegisterSetHandler` function shall be provided to give the possibility to register a `SetHandler`.

```
ara::core::Result<void> RegisterSetHandler(
        std::function<ara::core::Future<FieldType>(
                const FieldType& value)> setHandler);
```

⌋*(RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127)*

[SWS_CM_11362]{DRAFT} **Execution Context for registering Setters** ⌈For the `RegisterSetHandler` method described in [SWS_CM_00116] a second overload with an additional input parameter shall be provided. This parameter shall provide an executioner object in which any asynchronous computation spawn by `RegisterSetHandler` shall be invoked. The minimum behavior of the Execution Context is defined in [SWS_CM_11364].

```
template<typename ExecutorT>
ara::core::Result<void> RegisterSetHandler(
  std::function<ara::core::Future<FieldType>(
          const FieldType& value)> setHandler, ExecutorT&& executor);
```

For the first overload without an execution context argument an implementation defined default execution context (like in previous AUTOSAR releases) shall be used.⌋ *(RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127, RS_AP_00138)*

[SWS_CM_11363]{DRAFT} **Error behaviour of provided Execution Context for registering Setters** ⌈In case a `RegisterGetHandler()` cannot be executed with the provided executor (e.g. because of resource problem) a `ComErrc::kCouldNotExecute` error shall be raised in all cases.⌋*(RS_CM_00211, RS_CM_00212, RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127)*

[SWS_CM_00117]{DRAFT} **Existence of the RegisterSetHandler method** ⌈The existence of `RegisterSetHandler` as part of the `Field` class shall be controlled by `Field.hasSetter`.⌋*(RS_CM_00218, RS_AP_00114)*

[SWS_CM_00015]{DRAFT} **Re-entrancy and thread-safety - `RegisterSetHandler`** ⌈`RegisterSetHandler` shall be re-entrant and thread-safe for *different* `Field` class instances. When called re-entrant or concurrently on the same `Field` class instance, the behavior is undefined.⌋*(RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127)*

[SWS_CM_00119]{DRAFT} **Update Function** ⌈Inside the specific `Field` class belonging to the specific `ServiceSkeleton` class an `Update` function shall be provided to initiate the transmission of updated field data to the subscribers. See [SWS_CM_00162] for the required behavior. The Update method shall look as follows:

```
ara::core::Result<void> Field::Update(const FieldType &value);
```

It shall return void if the connection is successful, or an `ara::core::ErrorCode` from the `ara::com::ComErrorDomain` indicating the error if not successful (see chapter 8.1.2.6 Error Types).

The following errors from the ara::core::ComErrorDomain are possible:

- **kServiceNotOffered:** Service not offered.
- **kCommunicationLinkError:** Communication link is broken.
- **kCommunicationStackError:** Communication Stack Error, e.g. network stack, network binding, or communication framework reports an error

The Update function shall also update the field's internal value, if:

- If `Field.hasGetter` is true (according to [SWS_CM_00132] and
- no custom Getter has been registered

An update notification shall be sent, if hasNotification is true (in accordance to [SWS_CM_00120].⌋*(RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

[SWS_CM_00016]{DRAFT} **Re-entrancy and thread-safety - `Update`** ⌈`Update` shall be re-entrant and thread-safe for *different* `Field` class instances. When called re-entrant or concurrently on the same `Field` class instance, the behavior is undefined.⌋*(RS_CM_00218, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

8.1.3.10 Find service

For the functional description of the find service API, see chapter 7.10.6.

The Communication Management shall provide `FindService` methods as part of the `ServiceProxy` class to enable applications to find services. To support event-based and time-triggered systems the `FindService` methods shall be provided in a handler registration and a immediately returned request style.

[SWS_CM_00122]{DRAFT} **Find service with immediately returned request using Instance ID** ⌈The `FindService` method of the `ServiceProxy` class with immediately returned request takes an instance ID qualifying the wanted `instance` of the service as input parameter.
It shall return a result struct encapsulating a container of handles for all matching service instances, or an `ara::core::ErrorCode` from the `ara::com::ComErrorDomain` indicating the error if not successful.

There is one `FindService` method for using a specified `InstanceIdentifier`.

```
static ara::core::Result<ara::com::ServiceHandleContainer<
    <ProxyClassName>::HandleType>>
  FindService(ara::com::InstanceIdentifier instance);
```

where `<ProxyClassName>` is the name of the `ServiceProxy` class as defined in [SWS_CM_00004].

The following errors from `ara::com::ComErrorDomain` are possible:

- `kNetworkBindingFailure`: Local failure has been detected by the network binding.
- `kGrantEnforcementError`: Request was refused by `Grant` enforcement layer.
- `kPeerIsUnreachable`: Transport Layer Security handshake failed.

`InstanceIdentifier` validation errors or allocation failures of the `ServiceHandleContainer` should be treated as `Violation`s. ([SWS_CORE_00003], [SWS_CORE_00005])⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00120, RS_AP_00121, RS_AP_00119)*

For the definition of the types used in the `FindService` signature, see:

- [SWS_CM_00304] for `ServiceHandleContainer`,
- [SWS_CM_00312] for `HandleType`,
- [SWS_CM_00302] for `InstanceIdentifier`.

[SWS_CM_00622]{DRAFT} **Find service with immediately returned request using Instance Specifier** ⌈The `FindService` method of the `ServiceProxy` class with immediately returned request takes an instance Specifier qualifying the wanted Abstract Network Binding for the `instance`.
It shall return a result struct encapsulating a container of handles for all matching service instances, or an `ara::core::ErrorCode` from the `ara::com::ComErrorDomain` indicating the error if not successful.

```
static ara::core::Result<ara::com::ServiceHandleContainer<
    <ProxyClassName>::HandleType>>
  FindService(ara::core::InstanceSpecifier instance);
```

where `<ProxyClassName>` is the name of the `ServiceProxy` class as defined in [SWS_CM_00004].

The following errors from `ara::com::ComErrorDomain` are possible:

- `kNetworkBindingFailure`: Local failure has been detected by the network binding.

- kGrantEnforcementError: Request was refused by Grant enforcement layer.
- kPeerIsUnreachable: Transport Layer Security handshake failed.

InstanceSpecifier validation errors, or allocation failures of the ServiceHandleContainer should be treated as Violations. ([SWS_CORE_00003], [SWS_CORE_00005])⌋(*RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00119, RS_AP_00127, RS_AP_00137*)

For the definition of the types used in the FindService signature, see:

- [SWS_CM_00304] for ServiceHandleContainer,
- [SWS_CM_00312] for HandleType,
- [SWS_CORE_08001] for InstanceSpecifier.

[SWS_CM_00018]{DRAFT} **Re-entrancy and thread-safety - FindService** ⌈FindService is neither re-entrant nor thread-safe. When called re-entrant or concurrently, the behavior is undefined.⌋(*RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00119, RS_AP_00127, RS_AP_00137*)

[SWS_CM_00123]{DRAFT} **Find service with handler registration using Instance ID** ⌈The StartFindService method of the ServiceProxy class with handler registration takes as input parameters a FindServiceHandler, fitting for the corresponding ServiceProxy class which gets called upon detection of a matching service, and an instance ID qualifying the wanted instance of the service. The return value is a result struct encapsulating a FindServiceHandle for this search/find request, or an ara::core::ErrorCode from the ara::com::ComErrorDomain indicating the error if not successful. The FindServiceHandle is needed to stop the service availability monitoring and related firing of the given handler.

There is one StartFindService method for using a specified InstanceIdentifier.

```
static ara::core::Result<ara::com::FindServiceHandle> StartFindService(
  ara::com::FindServiceHandler<<ProxyClassName>::HandleType> handler,
  ara::com::InstanceIdentifier instance);
```

where <ProxyClassName> is the name of the ServiceProxy class as defined in [SWS_CM_00004].

The following errors from ara::com::ComErrorDomain are possible:

- kNetworkBindingFailure: Local failure has been detected by the network binding.
- kGrantEnforcementError: Request was refused by Grant enforcement layer.
- kPeerIsUnreachable: Transport Layer Security handshake failed.

`InstanceIdentifier` validation errors or allocation failures of the `ServiceHandleContainer` should be treated as `Violation`s. ([SWS_CORE_00003], [SWS_CORE_00005])

⌋(*RS_CM_00102*, *RS_AP_00114*, *RS_AP_00115*, *RS_AP_00120*, *RS_AP_00121*, *RS_AP_00119*)

For the definition of the types used in the `StartFindService` signature, see:

- [SWS_CM_00303] for `FindServiceHandle`,
- [SWS_CM_00383] for `FindServiceHandler`,
- [SWS_CM_00312] for `HandleType`,
- [SWS_CM_00302] for `InstanceIdentifier`.

Note: StartFindService is an asynchronous indication of availability and not to be abused for liveness monitoring.

[SWS_CM_11352]{DRAFT} **Execution Context for finding service with handler registration using Instance ID** ⌈For the `StartFindService` method described in [SWS_CM_00123] a second overload with an additional input parameter shall be provided. This parameter shall provide an executioner object in which any asynchronous computation spawn by `StartFindService` shall be invoked. The minimum behavior of the Execution Context is defined in [SWS_CM_11364].

```
template<typename ExecutorT>
static ara::com::FindServiceHandle StartFindService(
  ara::com::FindServiceHandler<<ProxyClassName>::HandleType> handler,
  ara::com::InstanceIdentifier instance, ExecutorT&& executor);
```

For the first overload without an execution context argument an implementation defined default execution context (like in previous AUTOSAR releases) shall be used.⌋ (*RS_CM_00102*, *RS_AP_00114*, *RS_AP_00115*, *RS_AP_00120*, *RS_AP_00121*, *RS_AP_00119*)

[SWS_CM_11353]{DRAFT} **Error behavior of provided Execution Context for finding service with handler registration using Instance ID** ⌈In case a `StartFindService()` cannot be executed with the provided executor (e.g. because of resource problem) an `ComErrc::kCouldNotExecute` error shall be raised in all cases.⌋(*RS_CM_00211*, *RS_CM_00212*, *RS_CM_00213*, *RS_CM_00214*, *RS_AP_00114*, *RS_AP_00119*, *RS_AP_00127*)

[SWS_CM_00623]{DRAFT} **Find service with handler registration using Instance Specifier** ⌈The `StartFindService` method of the `ServiceProxy` class with handler registration takes as input parameters a `FindServiceHandler`, fitting for the corresponding `ServiceProxy` class which gets called upon detection of a matching service, and an instance Specifier qualifying the wanted Abstract Network Binding of the `instance` of the service. The return value is a result struct encapsulating a `FindServiceHandle` for this search/find request, or an `ara::core::ErrorCode` from

the ara::com::ComErrorDomain indicating the error if not successful. The FindServiceHandle is needed to stop the service availability monitoring and related firing of the given handler.

```
static ara::core::Result<ara::com::FindServiceHandle> StartFindService(
ara::com::FindServiceHandler<<ProxyClassName>::HandleType> handler,
ara::core::InstanceSpecifier instance);
```

where <ProxyClassName> is the name of the ServiceProxy class as defined in [SWS_CM_00004].

The following errors from ara::com::ComErrorDomain are possible:

- kNetworkBindingFailure: Local failure has been detected by the network binding.
- kGrantEnforcementError: Request was refused by Grant enforcement layer.
- kPeerIsUnreachable: Transport Layer Security handshake failed.

InstanceSpecifier validation errors, or allocation failures of the ServiceHandleContainer should be treated as Violations. ([SWS_CORE_00003], [SWS_CORE_00005])⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00119, RS_AP_00127, RS_AP_00137)*

For the definition of the types used in the StartFindService signature, see:

- [SWS_CM_00303] for FindServiceHandle,
- [SWS_CM_00383] for FindServiceHandler,
- [SWS_CM_00312] for HandleType,
- [SWS_CORE_08001] for InstanceSpecifier.

Note: StartFindService is an asynchronous indication of availability and not to be abused for liveness monitoring.

[SWS_CM_00019]{DRAFT} **Re-entrancy and thread-safety - StartFindService** ⌈StartFindService is neither re-entrant nor thread-safe. When called re-entrant or concurrently, the behavior is undefined.⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00115, RS_AP_00121, RS_AP_00119, RS_AP_00127, RS_AP_00137)*

[SWS_CM_00125]{DRAFT} **Stop find service** ⌈To stop receiving further notifications the ServiceProxy class shall provide a StopFindService method. The FindServiceHandle returned by the FindService method with handler registration has to be provided as input parameter.

```
void StopFindService(ara::com::FindServiceHandle handle)
```

⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

See [SWS_CM_00303] for the type definition of FindServiceHandle.

[SWS_CM_00020]{DRAFT} **Re-entrancy and thread-safety - `StopFindService`** ⌈`StopFindService` shall be re-entrant and thread-safe for *different* `ara::com::FindServiceHandle`s. When called re-entrant or concurrently with the same `ara::com::FindServiceHandle`, the behavior is undefined.⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

8.1.3.11 Service proxy creation

[SWS_CM_00131]{DRAFT} **Creation of service proxy** ⌈The Communication Management shall provide a constructor for each specific `ServiceProxy` class taking a `handle` returned by any `FindService` method of the `ServiceProxy` class to get a valid `ServiceProxy` based on the handles returned by `FindService`.

```
explicit ServiceProxy::ServiceProxy(const HandleType &handle);
```

⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00121, RS_AP_00145)*

[SWS_CM_10438]{DRAFT} **Exception-less creation of service proxy** ⌈The Communication Management shall provide a non-throwing constructor for each specific `ServiceProxy` class using the Named Constructor idiom according to [SWS_CM_11326]. The Named Constructor shall be called Create() and shall take a `handle` returned by any `FindService` method of the `ServiceProxy` class as argument.

```
static ara::core::Result<ServiceProxy> Create(
          const HandleType &handle) noexcept;
```

In case the handle returned from FindService is corrupt, an error code kErroneousFileHandle shall be returned in the Result.

In case of a `Grant` enforcement failure, an error code `ComErrc::kGrantEnforcementError` shall be returned in the Result. See [SWS_CM_90006].

⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00121, RS_AP_00139, RS_AP_00128, RS_AP_00132, RS_AP_00127, RS_AP_00139, RS_AP_00145)*

[SWS_CM_10383]{DRAFT} **GetHandle function to return the proxy instance creation handle** ⌈The Communication Management shall provide a `GetHandle` method for each specific `ServiceProxy` class to get the handle from which the `ServiceProxy` instance has been created.

```
HandleType ServiceProxy::GetHandle() const;
```

⌋*(RS_CM_00107, RS_AP_00114, RS_AP_00119)*

See [SWS_CM_00312] for the type definition of `HandleType`.

[SWS_CM_00021]{DRAFT} **Re-entrancy and thread-safety - `GetHandle`** ⌈`GetHandle` shall be re-entrant and thread-safe irrespective of the `ServiceProxy` class instance. - i.e. `GetHandle` shall be re-entrant and thread-safe for

the same `ServiceProxy` class instance and for *different* `ServiceProxy` class instances.⌋*(RS_CM_00107, RS_AP_00114, RS_AP_00119)*

[SWS_CM_00136]{DRAFT} **Copy semantics of service proxy class** ⌈The Communication Management shall disable the generation of the copy constructor and the copy assignment operator for each specific `ServiceProxy` class.

```
ServiceProxy(const ServiceProxy&) = delete;
ServiceProxy& operator=(const ServiceProxy&) = delete;
```

⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00145, RS_AP_00147)*

[SWS_CM_00137]{DRAFT} **Move semantics of service proxy class** ⌈The Communication Management shall provide the possibility to move construct and move assign a `ServiceProxy` instance from another instance.

```
ServiceProxy(ServiceProxy &&);
ServiceProxy& operator=(ServiceProxy &&);
```

⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00145, RS_AP_00147)*

[SWS_CM_10491]{DRAFT} **Re-establishing service connection** ⌈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.⌋*(RS_CM_00102, RS_CM_00107)*

8.1.3.12 Service proxy destruction

[SWS_CM_10446]{DRAFT} **Destruction of service proxy** ⌈The destructor of each specific `ServiceProxy` class shall destroy the `Promise` instances corresponding to the `Future` instances returned by the function call operator (`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]) by explicitly or implicitly invoking the destructor of the `Promise` (see [SWS_CORE_00349]). This in turn will make the corresponding `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 `broken_promise` (see [SWS_CORE_00400]).⌋*(RS_CM_00102, RS_AP_00114, RS_AP_00127, RS_AP_00145)*

8.1.3.13 Service event subscription

[SWS_CM_00141] Method to subscribe to a service event ⌈Inside the specific `Event` class belonging to the specific `ServiceProxy` class a `Subscribe` method shall be provided to start subscription of the corresponding event. As input parameter the `cacheSize` of the subscription needs to be specified.

```
ara::core:Result<void> Event::Subscribe(
        std::size_t maxSampleCount
);
```

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, Subscribe() shall return silently without any action.

If the Event is already subscribed to at the time of the call, and the provided maxSampleCount value is different from the value for the current subscription, Subscribe() shall return the error code ComErrc::kMaxSampleCountNotRealizable in the Result.⌋*(RS_CM_00103, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00700]{DRAFT} **Ensure memory allocation of maxSampleCount samples** ⌈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.⌋*(RS_CM_00103, RS_AP_00114)*

[SWS_CM_00022]{DRAFT} **Re-entrancy and thread-safety - Subscribe** ⌈Subscribe shall be re-entrant and thread-safe for *different* Event class instances. When called re-entrant or concurrently on the same Event class instance, the behavior is undefined.⌋*(RS_CM_00103, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00151] Method to unsubscribe from a service event ⌈Inside the specific Event class belonging to the specific ServiceProxy class a Unsubscribe method shall be provided to allow for unsubscribing from previously subscribed events.

```
void Event::Unsubscribe();
```

If the Event is not subscribed to at the time of the call, Unsubscribe() shall return silently without any action.⌋*(RS_CM_00104, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00023]{DRAFT} **Re-entrancy and thread-safety - Unsubscribe** ⌈Unsubscribe shall be re-entrant and thread-safe for *different* Event class instances. When called re-entrant or concurrently on the same Event class instance, the behavior is undefined.⌋*(RS_CM_00104, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00316] Query Subscription State ⌈The Communication Management shall provide an API GetSubscriptionState which returns the subscription state of an event. The conditions for the Subscription state being returned by GetSubscriptionState shall be the same as for the SubscriptionStateChangeHandler described in [SWS_CM_00311], [SWS_CM_00313] and [SWS_CM_00314].

```
1 ara::com::SubscriptionState GetSubscriptionState() const;
```

⌋*(RS_CM_00106, RS_AP_00114, RS_AP_00115, RS_AP_00120, RS_AP_00119)*

[SWS_CM_00024]{DRAFT} **Re-entrancy and thread-safety - GetSubscriptionState** ⌈GetSubscriptionState shall be re-entrant and thread-safe for *different* Event class instances. When called re-entrant or concurrently on the same Event

class instance, the behavior is undefined.⌋*(RS_CM_00106, RS_AP_00114, RS_AP_00115, RS_AP_00120, RS_AP_00119)*

[SWS_CM_00333] Set Subscription State change handler ⌈The Communication Management shall provide an API `SetSubscriptionStateChangeHandler` to give the possibility to set a subscription state change handler. This handler 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.

```
ara::core::Result<void> SetSubscriptionStateChangeHandler(ara::com::
  SubscriptionStateChangeHandler handler);
```

⌋*(RS_CM_00106, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

[SWS_CM_11354]{DRAFT} **Execution Context for setting Subscription State change handler** ⌈For the `SetSubscriptionStateChangeHandler` method described in [SWS_CM_00333] a second overload with an additional input parameter shall be provided. This parameter shall provide an executioner object in which any asynchronous computation spawn by `SetSubscriptionStateChangeHandler` shall be invoked. The minimum behavior of the Execution Context is defined in [SWS_CM_11364].

```
template<typename ExecutorT>
ara::core::Result<void> SetSubscriptionStateChangeHandler(
  ara::com::SubscriptionStateChangeHandler handler, ExecutorT&& executor);
```

For the first overload without an execution context argument an implementation defined default execution context (like in previous AUTOSAR releases) shall be used.⌋ *(RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127, RS_AP_00138)*

[SWS_CM_11355]{DRAFT} **Error behaviour of provided Execution Context for setting Subscription State change handler** ⌈In case a `SetSubscriptionStateChangeHandler()` cannot be executed with the provided executor (e.g. because of resource problem) an `ComErrc::kCouldNotExecute` error shall be raised in all cases.⌋*(RS_CM_00211, RS_CM_00212, RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127)*

[SWS_CM_00025]{DRAFT} **Re-entrancy and thread-safety - `SetSubscriptionStateChangeHandler`** ⌈`SetSubscriptionStateChangeHandler` shall be re-entrant and thread-safe for *different* `Event` class instances. When called re-entrant or concurrently on the same `Event` class instance, the behavior is undefined.⌋*(RS_CM_00106, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

[SWS_CM_00334] Unset Subscription State change handler ⌈The Communication Management shall provide an API `UnsetSubscriptionStateChangeHandler` to give the possibility to unset the subscription state change handler.

```
void UnsetSubscriptionStateChangeHandler();
```

⌋*(RS_CM_00106, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00026]{DRAFT} **Re-entrancy and thread-safety - `UnsetSubscriptionStateChangeHandler`** ⌈`UnsetSubscriptionStateChangeHandler` shall be re-entrant and thread-safe for *different* `Event` class instances. When called re-entrant or concurrently on the same `Event` class instance, the behavior is undefined.⌋ *(RS_CM_00106, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00313] Call SubscriptionStateChangeHandler with kSubscriptionPending ⌈The Communication Management shall call the `SubscriptionStateChangeHandler` 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)

⌋*(RS_CM_00103, RS_CM_00104, RS_CM_00106, RS_CM_00107, RS_AP_00114)*

Note: Method Calls may lead to a `kServiceNotAvailable` error [SWS_CM_11264] at that time.

[SWS_CM_00314] Call SubscriptionStateChangeHandler with kSubscribed ⌈The Communication Management shall call the `SubscriptionStateChangeHandler` 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)

⌋*(RS_CM_00103, RS_CM_00104, RS_CM_00106, RS_CM_00107, RS_AP_00114)*

[SWS_CM_00315] Re-establishing an active subscription ⌈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.⌋*(RS_CM_00103, RS_CM_00104, RS_CM_00106, RS_CM_00107, RS_AP_00114)*

8.1.3.14 Receive event

Inside the specific `Event` class belonging to the specific `ServiceProxy` class, a `GetNewSamples` and a `GetFreeSampleCount` method shall be provided to allow for access of received events.

[SWS_CM_00701] Method to update the event cache ⌈The Communication Management shall provide an `GetNewSamples` method as part of the `Event` class to update the event cache with the meanwhile received data samples. As input parameters the `GetNewSamples` method expects a `Callable f` and allows to specify a `maxNumberOfSamples` to restrict the number of received data samples being processed in this call.

```
template <typename F>
ara::core::Result<std::size_t> GetNewSamples(
        F&& f,
        std::size_t maxNumberOfSamples =
                std::numeric_limits<std::size_t>::max());
```

⌋(*RS_CM_00202*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00121*, *RS_AP_00139*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00139*)

[SWS_CM_00702] Signature of Callable f ⌈The user provided `Callable f` has to comply with the following signature:

```
void(ara::com::SamplePtr<SampleType const>)
```

For the definition of the types used in the signature of `f`, see:

- [SWS_CM_00306] for `SamplePtr`.

⌋(*RS_CM_00202*, *RS_AP_00114*)

[SWS_CM_00703] Sequence of actions in GetNewSamples ⌈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 `SamplePtr` (including `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 `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.
- `size_t` indicating the number of data samples passed to `f` in the context of the call.

⌋(*RS_CM_00202*, *RS_AP_00114*)

[SWS_CM_00704]{DRAFT} **Return Value** ⌈The returned `ara::core::Result` contains a `ara::core::ErrorCode` (see [SWS_CORE_00501]) where the error domain is set to `ara::com::ComErrorDomain` with the value `kMaxSamplesExceeded` indicating, that applications `SamplePtr`s count has been exceeded. This means that all `SamplePtr`s are currently held by the application and no more samples can be delivered.⌋*(RS_CM_00202, RS_AP_00114, RS_AP_00119, RS_AP_00127)*

Note: This means that `maxSampleCount`, which is given in the `Subscribe()` method is exceeded

[SWS_CM_11358]{DRAFT} **Execution Context to update the event cache** ⌈For the `GetNewSamples` method described in [SWS_CM_00701] a second overload with an additional input parameter shall be provided. This parameter shall provide an executioner object in which any asynchronous computation spawn by `GetNewSamples` shall be invoked. The minimum behavior of the Execution Context is defined in [SWS_CM_11364].

```
template<typename ExecutorT>
ara::core::Result<std::size_t> GetNewSamples(
        F&& f,
        std::size_t maxNumberOfSamples =
                std::numeric_limits<std::size_t>::max(),
        ExecutorT&& executor);
```

For the first overload without an execution context argument an implementation defined default execution context (like in previous AUTOSAR releases) shall be used.⌋ *(RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127, RS_AP_00138)*

[SWS_CM_11359]{DRAFT} **Error behaviour of provided Execution Context to update the event cache** ⌈In case a `GetNewSamples()` cannot be executed with the provided executor (e.g. because of resource problem) an `ComErrc::kCouldNotExecute` error shall be raised in all cases.⌋*(RS_CM_00211, RS_CM_00212, RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127)*

[SWS_CM_00714] Re-entrancy and thread-safety - `GetNewSamples` ⌈`GetNewSamples` shall be re-entrant and thread-safe for *different* `Event` class instances. When called re-entrant or concurrently on the same `Event` class instance, the behavior is undefined. (If required, the application shall implement the locks).⌋ *(RS_CM_00202, RS_AP_00114)*

For the E2E-protected `event`s, after updating the event cache via the `GetNewSamples` method, and before accessing the `SamplePtr`s, the current `Result` needs to be retrieved by calling the `GetResult` method.

[SWS_CM_90424] Provide E2E Result ⌈Inside the specific E2E-protected `Event`s belonging to the specific `ServiceProxy` class, the method `GetResult` shall be provided.

```
const ara::com::e2e::Result GetResult() const;
```

⌋(*RS_E2E_08534*, *RS_AP_00114*, *RS_AP_00115*)

[SWS_CM_00705] Query Free Sample Slots ⌈The Communication Management shall provide a `GetFreeSampleCount` method as part of the `Event` class to query the number of free/unused slots for event sample data.

```
std::size_t GetFreeSampleCount() const noexcept;
```

⌋(*RS_CM_00202*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00139*, *RS_AP_00128*, *RS_AP_00132*, *RS_AP_00127*, *RS_AP_00139*)

[SWS_CM_00706] Return Value of GetFreeSampleCount ⌈The returned `size_t` indicates the number of free/unused slots for event sample data in the local cache.⌋ (*RS_CM_00202*, *RS_AP_00114*, *RS_AP_00119*, *RS_AP_00139*, *RS_AP_00128*, *RS_AP_00127*)

[SWS_CM_00707] Calculation of Free Sample Count ⌈

- 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 `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 `SamplePtr` instance created from this `Event` class instance shall lead to an increment of count of free samples.

⌋(*RS_CM_00202*, *RS_AP_00114*)

[SWS_CM_00027]{DRAFT} **Re-entrancy and thread-safety - `GetFreeSampleCount`** ⌈`GetFreeSampleCount` shall be re-entrant and thread-safe irrespective of the `Event` class instance i.e. `GetFreeSampleCount` shall be re-entrant and thread-safe for the same `Event` class instance and for *different* `Event` class instances.⌋(*RS_CM_00202*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00139*, *RS_AP_00128*, *RS_AP_00132*, *RS_AP_00127*, *RS_AP_00139*)

8.1.3.15 Receive event by getting triggered

For the functional description of the receive event by getting triggered API, see chapter 7.10.7.2.

[SWS_CM_00181]{DRAFT} **Enable service event trigger** ⌈To enable that applications get triggered upon receiving of an event inside the specific `Event` class belonging to the specific `ServiceProxy` class a `SetReceiveHandler` method shall be provided to allow for specifying the function to call upon event arrival. Therefore, it takes as input parameter `handler` a pointer to the respective function.

```
ara::core::Result<void> Event::SetReceiveHandler(
  ara::com::EventReceiveHandler handler);
```

The EventReceiveHandler constitutes a function without parameters and has to use the GetNewSamples method of the specific Event class to access the retrieved event data. See [SWS_CM_00309] for its definition.

In case SetReceiveHandler() fails, ComErrc::kSetHandlerNotSet shall be returned in the Result.⌋*(RS_CM_00203, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

[SWS_CM_11356]{DRAFT} **Execution Context for enabling service event trigger** ⌈For the SetReceiveHandler method described in [SWS_CM_00181] a second overload with an additional input parameter shall be provided. This parameter shall provide an executioner object in which any asynchronous computation spawn by SetReceiveHandler shall be invoked. The minimum behavior of the Execution Context is defined in [SWS_CM_11364].

```
template<typename ExecutorT>
ara::core::Result<void> Event::SetReceiveHandler(
  ara::com::EventReceiveHandler handler, ExecutorT&& executor);
```

For the first overload without an execution context argument an implementation defined default execution context (like in previous AUTOSAR releases) shall be used.⌋*(RS_CM_00211, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127, RS_AP_00138)*

[SWS_CM_11357]{DRAFT} **Error behaviour of provided Execution Context for enabling service event trigger** ⌈In case a SetReceiveHandler() cannot be executed with the provided executor (e.g. because of resource problem) an ComErrc::kCouldNotExecute error shall be raised in all cases.⌋*(RS_CM_00211, RS_CM_00212, RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127)*

[SWS_CM_00028]{DRAFT} **Re-entrancy and thread-safety - SetReceiveHandler** ⌈SetReceiveHandler shall be re-entrant and thread-safe for *different* Event class instances. When called re-entrant or concurrently on the same Event class instance, the behavior is undefined.⌋*(RS_CM_00203, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

[SWS_CM_00183]{DRAFT} **Disable service event trigger** ⌈To disable the triggering of the application upon receiving of an event inside the specific Event class belonging to the specific ServiceProxy class an UnsetReceiveHandler method shall be provided to allow for disabling of triggering the application.

```
ara::core::Result<void> Event::UnsetReceiveHandler();
```

In case UnsetReceiveHandler() fails, ComErrc::kUnsetFailure shall be returned in the Result.

⌋*(RS_CM_00203, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00029]{DRAFT} **Re-entrancy and thread-safety - `UnsetReceiveHandler`** ⌈`UnsetReceiveHandler` shall be re-entrant and thread-safe for *different* `Event` class instances. When called re-entrant or concurrently on the same `Event` class instance, the behavior is undefined.⌋*(RS_CM_00203, RS_AP_00114, RS_AP_00120)*

8.1.3.16 Service Trigger subscription

[SWS_CM_00723]{DRAFT} **Method to subscribe to a service trigger** ⌈Inside the specific `Trigger` class belonging to the specific `ServiceProxy` class a `Subscribe` method shall be provided to start subscription of the corresponding trigger.

```
ara::core::Result<void> Trigger::Subscribe();
```

If the `Trigger` is already subscribed to at the time of the call, and `Subscribe()` is invoked, it shall return silently without any action.⌋*(RS_CM_00103, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00724]{DRAFT} **Re-entrancy and thread-safety - `Subscribe`** ⌈`Subscribe` shall be re-entrant and thread-safe for different `Trigger` class instances. When called re-entrant or concurrently on the same `Trigger` class instance, the behavior is undefined.⌋*(RS_CM_00103, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00810]{DRAFT} **Method to unsubscribe from a service trigger** ⌈Inside the specific `Trigger` class belonging to the specific `ServiceProxy` class a `Unsubscribe` method shall be provided to allow for unsubscribing from previously subscribed triggers.

```
void Trigger::Unsubscribe();
```

If the `Trigger` is not subscribed to at the time of the call, `Unsubscribe()` shall return silently without any action.⌋*(RS_CM_00104, RS_AP_00114, RS_AP_00120)*

[SWS_CM_00035]{DRAFT} **Re-entrancy and thread-safety - `Unsubscribe`** ⌈`Unsubscribe` shall be re-entrant and thread-safe for different `Trigger` class instances. When called re-entrant or concurrently on the same `Trigger` class instance, the behavior is undefined.⌋*(RS_CM_00104, RS_AP_00114, RS_AP_00120)*

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_00024] Reentrancy - `GetSubscriptionState`.
- [SWS_CM_00333] Set Subscription State change handler.
- [SWS_CM_11354] Execution Context for setting Subscription State change handler.

- [SWS_CM_11355] Error behavior of provided Execution Context for setting Subscription State change handler.
- [SWS_CM_00025] Reentrancy - `SetSubscriptionStateChangeHandler`.
- [SWS_CM_00334] Unset Subscription State change handler.
- [SWS_CM_00026] Reentrancy - `UnsetSubscriptionStateChangeHandler`.
- [SWS_CM_00313] Call SubscriptionStateChangeHandler with kSubscriptionPending.
- [SWS_CM_00314] Call SubscriptionStateChangeHandler with kSubscribed.
- [SWS_CM_00315] Re-establishing an active subscription.

8.1.3.17 Receive Trigger

Inside the specific `Trigger` class belonging to the specific `ServiceProxy` class, a `GetNewTriggers` method shall be provided to allow for access of received triggers.

[SWS_CM_00226]{DRAFT} **Method to update the trigger counter** ⌈The Communication Management shall provide an `GetNewTriggers` method as part of the `Trigger` class to update the trigger counter.

```
std::size_t GetNewTriggers();
```

⌋*(RS_CM_00202, RS_AP_00114, RS_AP_00120, RS_AP_00121, RS_AP_00139, RS_AP_00128, RS_AP_00127, RS_AP_00139)*

[SWS_CM_00227]{DRAFT} **Sequence of actions in GetNewTriggers** ⌈In the context of the `GetNewTriggers` call, the Communication Management shall get the number of triggers occurred since the last call of `GetNewTriggers`.⌋*(RS_CM_00202, RS_AP_00114)*

[SWS_CM_00228]{DRAFT} **Return Value** ⌈The returned `size_t` indicates the number of triggers occurred since the last call to `GetNewTriggers` (Zero value means that there is no new triggers received).⌋*(RS_CM_00202, RS_AP_00114, RS_AP_00119, RS_AP_00127)*

[SWS_CM_11251]{DRAFT} **Re-entrancy and thread-safety - `GetNewTriggers`** ⌈`GetNewTriggers` shall be re-entrant and thread-safe for different `Trigger` class instances. When called concurrently on the same `Trigger` class instance, the behavior is undefined.⌋*(RS_CM_00202, RS_AP_00114)*

8.1.3.18 Receive trigger by getting triggered

For the functional description of the receive trigger by getting triggered API, see [SWS_CM_00182].

[SWS_CM_00249]{DRAFT} **Enable service `Trigger` trigger** ⌈To enable that applications get triggered upon receiving of a trigger inside the specific `Trigger` class belonging to the specific `ServiceProxy` class a `SetReceiveHandler` method shall be provided to allow for specifying the function to call upon trigger arrival. Therefore, it takes as input parameter `handler` a pointer to the respective function.

```
ara::core::Result<void> Trigger::SetReceiveHandler(
  ara::com::TriggerReceiveHandler handler);
```

The `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 its definition. In case `SetReceiveHandler()` fails, `ComErrc::kSetHandlerNotSet` shall be returned in the Result.⌋*(RS_CM_00203, RS_AP_00114, RS_AP_00120, RS_AP_00121)*

The following specification are also valid for `Trigger`

- [SWS_CM_00028] Reentrancy - `SetReceiveHandler`
- [SWS_CM_00183] Disable service event trigger
- [SWS_CM_00029] Reentrancy - `UnsetReceiveHandler`

8.1.3.19 Call a service method

For the functional description of the call a service method API, see chapter 7.10.8.

[SWS_CM_00196]{DRAFT} **Initiate a method call** ⌈For each service method (i.e., `ServiceInterface.method` with `ClientServerOperation.fireAndForget` set to `false`) of a `ServiceInterface` a specific `Method` class named by the `ServiceInterface.method.shortName` shall be provided inside the specific `ServiceProxy` class of the `ServiceInterface`.
Within this `Method` class, a dedicated method `Output` type combining the possible output parameters (`ClientServerOperation.arguments` with `ArgumentDataPrototype.direction` set to `out` or `inout`) shall be provided.
Additionally the `operator()` shall be provided inside the specific `Method` class to allow the call of a method provided by a server.
As input parameters, the `operator()` shall take the respective input parameters (`ClientServerOperation.arguments` with `ArgumentDataPrototype.direction` set to `in` or `inout`) of the provided method.
The `operator()` shall return an `ara::core::Future` object wrapping the dedicated method `Output` type.

```
class Method {
  struct Output {
    TypeOutputParameter1 output1;
    TypeOutputParameter2 output2;
    ...
  };

  ara::core::Future<Output> operator()(
    TypeInputParameter1 input1,
    TypeInputParameter2 input2,
    ...
  );
};
```

⌋(*RS_CM_00212*, *RS_CM_00213*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00138*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00138*)

The method call according to [SWS_CM_00196] 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.

[SWS_CM_00032]{DRAFT} **Re-entrancy and thread-safety - `Method` call operator** ⌈`operator()` shall be re-entrant and thread-safe irrespective of the `Method` class instance i.e. `operator()` shall be re-entrant and thread-safe for the same `Method` class instance and for *different* `Method` class instances.⌋(*RS_CM_00212*, *RS_CM_00213*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00138*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00138*)

[SWS_CM_00194]{DRAFT} **Cancel the method call** ⌈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.⌋(*RS_CM_00212*, *RS_CM_00213*, *RS_AP_00114*, *RS_AP_00127*)

This is a mechanism on client side to tell the Communication Management software that the caller is not interested in the method result anymore. Cancellation of the method call is not propagated to the server side execution of the method.

[SWS_CM_00195]{DRAFT} **Retrieving results of the method call** ⌈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.⌋(*RS_CM_00212*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00138*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00139*)

[SWS_CM_00192]{DRAFT} **Synchronous behavior of method call** ⌈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()`, `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.⌋*(RS_CM_00212, RS_AP_00114, RS_AP_00120, RS_AP_00138, RS_AP_00128, RS_AP_00127, RS_AP_00138)*

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()`, `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 7.8.1) 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]{DRAFT} **Aborting method calls in case of locally detected failures** ⌈To notify the adaptive application about locally detected failures which

prevent an issued (remote) service method call from succeeding, the ara::com implementation shall make the Future returned by the function call operator (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_00347]) 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_11264]) and the value is set to kNetworkBindingFailure (see [SWS_CM_10432]) as an argument.⌋*(RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00119, RS_AP_00127)*

[SWS_CM_00193]{DRAFT} **Asynchronous behavior of method call with polling** ⌈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.⌋*(RS_CM_00213, RS_CM_00214, RS_AP_00114, RS_AP_00127)*

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]{DRAFT} **Asynchronous behavior of method call with notification** ⌈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.⌋*(RS_CM_00213, RS_CM_00215, RS_AP_00114, RS_AP_00127, RS_AP_00138)*

[SWS_CM_90435]{DRAFT} **Initiate a `Fire and Forget method` call** ⌈For each fire and forget service method (i.e., ServiceInterface.method with ClientServerOperation.fireAndForget set to true) of a ServiceInterface a specific FireAndForgetMethod class named by the ServiceInterface.method.shortName shall be provided inside the specific ServiceProxy class of the ServiceInterface.
Within this FireAndForgetMethod class, the operator() shall be provided to allow the call of a fire and forget method provided by a server.
As input parameters, the operator() shall take the respective input parameters (ClientServerOperation.arguments with ArgumentDataPrototype.direction set to in) of the provided fire and forget method.
The operator() shall not have return values.

```
class FireAndForgetMethod {
  void operator()(
    TypeInputParameter1 input1,
    TypeInputParameter2 input2,
```

```
    ...
  );
};
```

⌋(*RS_CM_00225*, *RS_AP_00114*, *RS_AP_00120*)

8.1.3.20 Get method for fields

[SWS_CM_00112]{DRAFT} **Method to get the value of a field** ⌈The Communication Management shall provide a `Get` method as part of the `Field` class to offer a service to request the current value of the service provider.

```
ara::core::Future<FieldType> Get();
```

⌋(*RS_CM_00218*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00138*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00138*)

[SWS_CM_00132]{DRAFT} **Existence of getter method** ⌈The existence of the `Get` method as part of the `Field` class shall be controlled by `Field.hasGetter`.⌋(*RS_CM_00218*, *RS_AP_00114*)

[SWS_CM_00030]{DRAFT} **Re-entrancy and thread-safety - `Get`** ⌈`Get` shall be re-entrant and thread-safe irrespective of the `Field` class instance i.e. `Get` shall be re-entrant and thread-safe for the same `Field` class instance and for *different* `Field` class instances.⌋(*RS_CM_00218*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00138*, *RS_AP_00128*, *RS_AP_00127*, *RS_AP_00138*)

8.1.3.21 Set method for fields

[SWS_CM_00113]{DRAFT} **Method to set the value of a field** ⌈The Communication Management shall provide a `Set` method as part of the `Field` class to offer a service to the applications to request the setting of a new value within the service provider.

```
ara::core::Future<FieldType> Set(const FieldType& value);
```

⌋(*RS_CM_00217*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00121*, *RS_AP_00138*, *RS_AP_00138*, *RS_AP_00127*, *RS_AP_00138*)

[SWS_CM_00133]{DRAFT} **Existence of the set method** ⌈The existence of the set method as part of the `Field` class shall be controlled by `Field.hasSetter`.⌋(*RS_CM_00218*, *RS_AP_00114*)

[SWS_CM_00031]{DRAFT} **Re-entrancy and thread-safety - `Set`** ⌈`Set` shall be re-entrant and thread-safe irrespective of the `Field` class instance i.e. `Set` shall be re-entrant and thread-safe for the same `Field` class instance and for *different* `Field` class instances.⌋(*RS_CM_00217*, *RS_AP_00114*, *RS_AP_00120*, *RS_AP_00121*, *RS_AP_00138*, *RS_AP_00138*, *RS_AP_00127*, *RS_AP_00138*)

8.1.3.22 Instance Specifier Translation

For the functional description of the Instance Specifier Translation API, see chapter 7.10.10.

[SWS_CM_00118]{DRAFT} ⌈

Kind:	function	
Symbol:	ResolveInstanceIDs(ara::core::InstanceSpecifier modelName)	
Scope:	namespace ara::com::runtime	
Syntax:	`ara::core::Result<ara::com::InstanceIdentifierContainer> ResolveInstanceIDs (ara::core::InstanceSpecifier modelName);`	
Parameters (in):	modelName	The instance specifier to be translated.
Return value:	ara::core::Result< ara::com::InstanceIdentifierContainer >	An Instance Identifier list if successful, otherwise an error code indicating the error
Errors:	ara::com::ComErrc::kInstanceIDCouldNotBeResolved	ResolveInstanceIDs() failed to resolve InstanceID from InstanceSpecifier, i.e. is not mapped correctly.
Header file:	#include "ara/com/runtime/runtime.h"	
Description:	Method Instance Specifier Translation. The Communication Management shall provide a ResolveInstanceIDs method to translate an InstanceSpecifier to an Instance Identifiers list. The size of the list could be 0, 1 or greater than 1 depending on the match.	

⌋(*RS_CM_00200*, *RS_AP_00114*, *RS_AP_00115*, *RS_AP_00120*, *RS_AP_00121*, *RS_AP_00119*, *RS_AP_00127*, *RS_AP_00137*)

For the definition of the types used in the `ResolveInstanceIDs` signature, see:

- [SWS_CM_00319] for `InstanceIdentifierContainer`,
- [SWS_CORE_08001] for `InstanceSpecifier`.

8.1.3.23 Raw Data Stream API

For the functional description of the Raw Data Stream API, see chapter 7.5.2.

[SWS_CM_10481] ⌈

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

⌋(*RS_CM_00410*, *RS_CM_00411*)

[SWS_CM_10482] ⌈

Kind:	function	
Symbol:	Create(const ara::core::InstanceSpecifier &instance)	
Scope:	class ara::com::raw::RawDataStreamClient	
Syntax:	`ara::core::Result<RawDataStreamClient> Create (const ara::core::InstanceSpecifier &instance) noexcept;`	
Parameters (in):	instance	The instance specifier for the instance.
Return value:	ara::core::Result< RawDataStream Client >	ara::core::Result<RawDataStreamClient> The Raw DataStreamClient object if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kConnection CreationFailed	Permission to create a connection is denied. (POSIX EACCES)
	ara::com::raw::RawErrc::kAddressNot Available	The specified address is not available from the local machine.
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Named exception-less constructor that takes an instance Specifier qualifying the wanted network binding and parameters for the instance. If Remote Unicast Credentials (TCP or UDP) are defined for the client, the constructor shall create an endpoint for the communication, and store the handle in the created RawDataStream Client object, to be used in the Read- and Write-operations for the RawDataStreamClient (for 1:1 use cases). If Multicast Credentials (UDP) are defined for the client, the constructor shall create an endpoint for the communication, bind and join the multicast address and port specified in the Multicast Credentials. For 1:N use cases this endpoint shall be used when RawDataStreamsClient.ReadData() is called, otherwise (for 1:1 use cases), the unicast endpoint shall be used for reading data.	

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

[SWS_CM_10483] ⌈

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

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

[SWS_CM_11303]{DRAFT} ⌈

Kind:	function
Symbol:	RawDataStreamClient(const RawDataStreamClient &)
Scope:	class ara::com::raw::RawDataStreamClient
Syntax:	`RawDataStreamClient (const RawDataStreamClient &)=delete;`

▽

△

Header file:	#include "ara/com/raw/raw_data_stream.h"
Description:	Copy constructor of the RawDataStreamClient - not allowed.

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

[SWS_CM_11304]{DRAFT} ⌈

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

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

[SWS_CM_11305]{DRAFT} ⌈

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

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

[SWS_CM_11306]{DRAFT} ⌈

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

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

[SWS_CM_10484] ⌈

Kind:	function	
Symbol:	Connect()	
Scope:	class ara::com::raw::RawDataStreamClient	
Syntax:	`ara::core::Result<void> Connect () noexcept;`	
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kConnection Refused	The connection was refused by target.
	ara::com::raw::RawErrc::kAddressNot Available	The specified address is not available from the local machine.
	ara::com::raw::RawErrc::kStream AlreadyConnected	The specified connection is already connected.
	ara::com::raw::RawErrc::kPeer Unreachable	The peer is unreachable by the network.
	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Sets up a unicast socket connection for the RawDataStream defined by the instance, and establishes a connection to the TCP server. In the case of UDP, no connection is established. Incoming and outgoing packets are restricted to the specified address. The socket endpoints and attributes are specified in the manifest which is accessed through the InstanceSpecifer provided in the constructor. If TLS security protocol is configured for the socket connection, the TLS/DTLS connection shall be initialized here.	

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

Note: For TLS/DTLS connection with Raw Data Streaming see also chapter 7.9.2.2.3.

[SWS_CM_11307]{DRAFT} ⌈

Kind:	function	
Symbol:	Connect(std::chrono::milliseconds timeout)	
Scope:	class ara::com::raw::RawDataStreamClient	
Syntax:	`ara::core::Result<void> Connect (std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	timeout	Timeout value for this operation.
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kConnection Refused	The connection was refused by target.
	ara::com::raw::RawErrc::kAddressNot Available	The specified address is not available from the local machine.
	ara::com::raw::RawErrc::k CommunicationTimeout	The connect operation timed out.
	ara::com::raw::RawErrc::kStream AlreadyConnected	The specified connection is already connected.
	ara::com::raw::RawErrc::kPeer Unreachable	The peer is unreachable by the network.

▽

△

	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Sets up a unicast socket connection for the RawDataStream defined by the instance, and establishes a connection to the TCP server. In the case of UDP, no connection is established. Incoming and outgoing packets are restricted to the specified address. The socket endpoints and attributes are specified in the manifest which is accessed through the InstanceSpecifer provided in the constructor. If TLS security protocol is configured for the socket connection, the TLS/DTLS connection shall be initialized here.	

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

[SWS_CM_10485] ⌈

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

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

[SWS_CM_10486] ⌈

Kind:	function	
Symbol:	ReadData(std::size_t maxLength)	
Scope:	class ara::com::raw::RawDataStreamClient	
Syntax:	`ara::core::Result<ReadDataResult> ReadData (std::size_t maxLength) noexcept;`	
Parameters (in):	maxLength	The requested number of bytes to read from the stream.
Return value:	ara::core::Result< ReadDataResult >	a struct of type ReadDataResult if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kStreamNot Connected	Trying to read from a stream without an established connection.
	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.

▽

△

Header file:	#include "ara/com/raw/raw_data_stream.h"
Description:	Requests to read a number of bytes of data from the socket connection for the RawDataStream defined by the instance. If Multicast Credentials are defined for the client, the data shall be read from the multicast socket created in the constructor (for 1:N use cases), otherwise the data shall be read from the unicast TCP socket connection set up in Connect() (for 1:1 TCP unicast use case), or the unicast UDP socket created in the constructor (for 1:1 UDP unicast use case). For efficiency, the zero-copy semantics of ara::com::SamplePtr is used, which means that the ownership of the allocated memory of the read data is transferred to the application in the Read DataResult.data value.

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

[SWS_CM_11309]{DRAFT} ⌈

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

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

[SWS_CM_10487] ⌈

Kind:	function
Symbol:	WriteData(ara::com::SamplePtr< std::uint8_t > data, std::size_t maxLength)
Scope:	class ara::com::raw::RawDataStreamClient
Syntax:	`ara::core::Result<std::size_t> WriteData (ara::com::SamplePtr< std::uint8_t > data, std::size_t maxLength) noexcept;`

▽

△

Parameters (in):	data	pointer to the byte array to send. A SamplePtr is used to get std::unique_ptr semantics.
	maxLength	The number of bytes to write to the stream.
Return value:	ara::core::Result< std::size_t >	the actual number of bytes written if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kStreamNot Connected	Trying to write to a stream without an established connection.
	ara::com::raw::RawErrc::kConnection ClosedByPeer	The established connection has been shut down during writing.
	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Requests to write of a number of bytes to the the socket connection for the RawDataStream defined by the instance (for 1:1 use cases). If Multicast Credentials are defined for the client reading of data, a single socket can be used for both multicast reading and unicast writing (for use case 1:N UDP + 1:1 UDP, Server sends data via multicast, and a client sends control data via unicast). For efficiency, the zero-copy semantics of ara::com::SamplePtr is used.	

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

[SWS_CM_11310]{DRAFT} ⌈

Kind:	function	
Symbol:	WriteData(ara::com::SamplePtr< std::uint8_t > data, std::size_t maxLength, std::chrono::milliseconds timeout)	
Scope:	class ara::com::raw::RawDataStreamClient	
Syntax:	`ara::core::Result<std::size_t> WriteData (ara::com::SamplePtr< std::uint8_t > data, std::size_t maxLength, std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	data	pointer to the byte array to send. A SamplePtr is used to get std::unique_ptr semantics.
	maxLength	The number of bytes to write to the stream.
	timeout	Timeout value for this operation.
Return value:	ara::core::Result< std::size_t >	the actual number of bytes written if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kStreamNot Connected	Trying to write to a stream without an established connection.
	ara::com::raw::RawErrc::k CommunicationTimeout	No data was written until the timeout expired.
	ara::com::raw::RawErrc::kConnection ClosedByPeer	The established connection has been shut down during writing.
	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.
Header file:	#include "ara/com/raw/raw_data_stream.h"	

▽

△

Description:	Requests to write a number of bytes to the the socket connection for the RawDataStream defined by the instance. If Multicast Credentials are defined for the client reading of data, a single socket can be used for both multicast reading and unicast writing (for use case 1:N UDP + 1:1 UDP, Server sends data via multicast, and a client sends control data via unicast). For efficiency, the zero-copy semantics of ara::com::SamplePtr is used.

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

[SWS_CM_11311]{DRAFT} ⌈

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

⌋(*RS_CM_00410*, *RS_CM_00411*)

[SWS_CM_11312]{DRAFT} ⌈

Kind:	function	
Symbol:	Create(const ara::core::InstanceSpecifier &instance)	
Scope:	class ara::com::raw::RawDataStreamServer	
Syntax:	`ara::core::Result<RawDataStreamServer> Create (const ara::core::InstanceSpecifier &instance) noexcept;`	
Parameters (in):	instance	The instance specifier for the instance.
Return value:	ara::core::Result< RawDataStream Server >	ara::core::Result<RawDataStreamServer> The Raw DataStreamServer object if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kConnection CreationFailed	Permission to create a connection is denied. (POSIX EACCES)
	ara::com::raw::RawErrc::kAddressNot Available	The specified address is not available from the local machine.
	ara::com::raw::RawErrc::kStream AlreadyConnected	The specified connection is already connected.
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Named exception-less constructor that takes an instance Specifier qualifying the wanted network credentials (UDP or TCP) for the instance. A socket shall be created and bound to the address and port specified in the local credentials. In case of TCP it shall also mark the socket as passive and listen for connections (for use case 1:1 TCP unicast). If Remote Unicast Credentials (UDP) are defined for the server, the constructor shall create an endpoint for the communication, and store the handle in the created RawDataStreamServer object, to be used in the Read and Write- operations for the RawDataStreamServer (for use case 1:1 UDP unicast). If Multicast Credentials (UDP) are defined for the server, the constructor shall create an endpoint for the remote communication, bind and join the multicast address and port specified in the MulticastCredentials. In this case, this endpoint shall be used when RawData StreamsServer.WriteData() is called (for 1:N use cases), otherwise the unicast endpoint shall be used for writing data (for 1:1 use cases).	

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

[SWS_CM_11313]{DRAFT} ⌈

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

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

[SWS_CM_11314]{DRAFT} ⌈

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

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

[SWS_CM_11315]{DRAFT} ⌈

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

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

[SWS_CM_11316]{DRAFT} ⌈

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

▽

△

Description:	Move constructor of the RawDataStreamServer.

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

[SWS_CM_11317]{DRAFT} ⌈

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

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

[SWS_CM_11318]{DRAFT} ⌈

Kind:	function	
Symbol:	WaitForConnection()	
Scope:	class ara::com::raw::RawDataStreamServer	
Syntax:	`ara::core::Result<void> WaitForConnection () noexcept;`	
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kConnection Aborted	The incoming connection was aborted by the network.
	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Enables the RawDataStreamServer instance for incoming connections. For TCP the constructor marks the socket as ready to accept connection requests from a client (see SWS_CM_11312), and WaitForConnection() waits to accept an incoming connection request. In the case of UDP, no connection is established, and the operation shall return with no action.	

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

[SWS_CM_11319]{DRAFT} ⌈

Kind:	function
Symbol:	WaitForConnection(std::chrono::milliseconds timeout)
Scope:	class ara::com::raw::RawDataStreamServer

▽

△

Syntax:	`ara::core::Result<void> WaitForConnection (std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	timeout	Timeout value for this operation.
Return value:	ara::core::Result< void >	void if successful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kCommunicationTimeout	The WaitForConnection operation timed out.
	ara::com::raw::RawErrc::kConnectionAborted	The incoming connection was aborted by the network.
	ara::com::raw::RawErrc::kInterruptedBySignal	The operation was interrupted by the system
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Enables the RawDataStreamServer instance for incoming connections. For TCP the constructor marks the socket as ready to accept connection requests from a client (see SWS_CM_11312), and WaitForConnection() waits to accept an incoming connection request. In the case of UDP, no connection is established, and the operation shall return with no action.	

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

[SWS_CM_11320]{DRAFT} ⌈

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

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

[SWS_CM_11322]{DRAFT} ⌈

Kind:	function
Symbol:	ReadData(std::size_t maxLength)
Scope:	class ara::com::raw::RawDataStreamServer
Syntax:	`ara::core::Result<ReadDataResult> ReadData (std::size_t maxLength) noexcept;`

▽

△

Parameters (in):	maxLength	The number of bytes to read from the stream.
Return value:	ara::core::Result< ReadDataResult >	a struct of type ReadDataResult if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kStreamNot Connected	Trying to read from a stream without an established connection.
	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Requests to read a number of bytes of data from the Unicast socket connection for the Raw DataStream defined by the instance. For efficiency, the zero-copy semantics of ara::com::SamplePtr is used, which means that the ownership of the allocated memory of the read data is transferred to the application in the Read DataResult.data value.	

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

[SWS_CM_11323]{DRAFT} ⌈

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

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

[SWS_CM_11324]{DRAFT} ⌈

Kind:	function
Symbol:	WriteData(ara::com::SamplePtr< std::uint8_t > data, std::size_t maxLength)
Scope:	class ara::com::raw::RawDataStreamServer

▽

△

Syntax:	`ara::core::Result<std::size_t> WriteData (ara::com::SamplePtr< std::uint8_t > data, std::size_t maxLength) noexcept;`	
Parameters (in):	data	pointer to the byte array to send. A SamplePtr is used to get std::unique_ptr semantics.
	maxLength	The number of bytes to write to the stream.
Return value:	ara::core::Result< std::size_t >	the actual number of bytes written if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kStreamNot Connected	Trying to write to a stream without an established connection.
	ara::com::raw::RawErrc::kConnection ClosedByPeer	The established connection has been shut down during writing.
	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.
Header file:	#include "ara/com/raw/raw_data_stream.h"	
Description:	Requests to write a number of bytes to the the socket connection for the RawDataStream defined by the instance. If Remote Multicast Credentials are defined for the server, the data shall be written to the multicast socket created in the constructor (for 1:N use cases). Otherwise in case of TCP, the data shall be written to the unicast socket connection set up in WaitForConnection() (for 1:1 TCP unicast use case). In case of UDP the data shall be written to the unicast socket created in the constructor (1:1 UDP unicast). For efficiency, the zero-copy semantics of ara::com::SamplePtr is used.	

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

[SWS_CM_11325]{DRAFT} ⌈

Kind:	function	
Symbol:	WriteData(ara::com::SamplePtr< std::uint8_t > data, std::size_t maxLength, std::chrono::milliseconds timeout)	
Scope:	class ara::com::raw::RawDataStreamServer	
Syntax:	`ara::core::Result<std::size_t> WriteData (ara::com::SamplePtr< std::uint8_t > data, std::size_t maxLength, std::chrono::milliseconds timeout) noexcept;`	
Parameters (in):	data	pointer to the byte array to send. A SamplePtr is used to get std::unique_ptr semantics.
	maxLength	The number of bytes to write to the stream.
	timeout	Parameter to assign a timeout for this operation.
Return value:	ara::core::Result< std::size_t >	the actual number of bytes written if succesful, otherwise an error code indicating the error.
Exception Safety:	noexcept	
Errors:	ara::com::raw::RawErrc::kStreamNot Connected	Trying to write to a stream without an established connection.
	ara::com::raw::RawErrc::k CommunicationTimeout	No data was written until the timeout expired.
	ara::com::raw::RawErrc::kConnection ClosedByPeer	The established connection has been shut down during writing.
	ara::com::raw::RawErrc::kInterrupted BySignal	The operation was interrupted by the system.
Header file:	#include "ara/com/raw/raw_data_stream.h"	

▽

△

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

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

For the definition of the types used in the `ReadData` and `WriteData` signature, see:

- [SWS_CM_00306] for `SamplePtr`

9 Service Interfaces

9.1 Service Interfaces

[SWS_CM_11280]{DRAFT} ⌈

Name	VerificationStatus
NameSpace	ara::com::secoc

Events	VerificationStatus
Type	`VerificationStatusContainer`

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11282]{DRAFT} ⌈

Name	VerificationStatusConfigurationByDataId
NameSpace	ara::com::secoc

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.	
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 k SecOcOverrideDropUntilLimit, kSecOcOverrideSkipUntilLimit or kSecOcOverride PassUntilLimit.
	Type	`uint8_t`
	Variation	
	Direction	IN

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11281]{DRAFT} ⌈

Name	VerificationStatusConfigurationByFreshnessId
NameSpace	ara::com::secoc

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.	
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 k SecOcOverrideDropUntilLimit, kSecOcOverrideSkipUntilLimit or kSecOcOverride PassUntilLimit.
	Type	`uint8_t`
	Variation	
	Direction	IN

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

9.2 Data Types

[SWS_CM_11285]{DRAFT} ⌈

Name	OverrideStatus	
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.

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11283]{DRAFT} ⌈

Name	VerificationStatusContainer
Kind	STRUCTURE
Subelements	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

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11284]{DRAFT} ⌈

Name	VerificationStatusResult	
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.

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

A Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document.

Class	***AbstractIamRemoteSubject*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SCREIAM			
Note	This abstract meta-class defines the proxy information about the remote node. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	IPSecIamRemoteSubject, IpIamRemoteSubject, TlsIamRemoteSubject			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.1: AbstractIamRemoteSubject

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

Table A.2: AbstractRawDataStreamEthernetCredentials

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. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Subclasses	*ProvidedApServiceInstance*, *RequiredApServiceInstance*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***AdaptivePlatformServiceInstance*** (abstract)			
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. **Tags:**atp.Status=draft
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 **Tags:**atp.Status=draft
secureCom Config	ServiceInterface ElementSecureCom Config	*	aggr	Configuration settings to secure the communication of ServiceInterface elements. **Tags:**atp.Status=draft
serviceInterface Deployment	ServiceInterface Deployment	0..1	ref	Reference to a ServiceInterfaceDeployment that identifies the ServiceInterface that is represented by the Service Instance. **Tags:**atp.Status=draft

Table A.3: AdaptivePlatformServiceInstance

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.Status=draft atp.recommendedPackage=ApplicationErrors			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
errorCode	Integer	1	attr	This attribute has the ability to specify the error code value within the enclosing AdaptivePlatformApplication Error. **Tags:**atp.Status=draft
errorDomain	ApApplicationError Domain	1	ref	This reference represents the error domain of the Ap ApplicationError. **Tags:**atp.Status=draft

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.Status=draft atp.recommendedPackage=ApplicationErrorDomains			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**ApApplicationErrorDomain**			
namespace (ordered)	SymbolProps	*	aggr	This aggregation defines the namespace of the Ap ApplicationErrorDomain **Tags:**atp.Status=draft
value	PositiveUnlimitedInteger	1	attr	This attribute identifies the error category. **Tags:**atp.Status=draft

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.Status=draft atp.recommendedPackage=ApplicationErrorSets			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
apApplication Error	ApApplicationError	*	ref	This reference represents the collection of ApApplication Error represented by the enclosing ApApplicationErrorSet **Tags:**atp.Status=draft

Table A.6: ApApplicationErrorSet

Class	**ApSomeipTransformationProps**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::SerializationProperties			
Note	SOME/IP serialization properties. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *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. **Tags:**atp.Status=draft
byteOrder	ByteOrderEnum	0..1	attr	Specifies the byte order of data in the serialized data stream. **Tags:**atp.Status=draft

▽

△

Class	**ApSomeipTransformationProps**			
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=draft
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. **Tags:**atp.Status=draft
session Handling	SOMEIPTransformer SessionHandlingEnum	0..1	attr	Defines whether the SOME/IP transformer shall use session handling for Sender/Receiver communication. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
stringEncoding	BaseTypeEncoding String	0..1	attr	Configures the encoding for SOME/IP serialization for the referenced dataPrototype in case of an String. **Tags:**atp.Status=draft

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*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dynamicArraySizeProfile	String	0..1	attr	Specifies the profile which the array will follow if it is a variable size array.
element	ApplicationArrayElement	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			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.9: ApplicationDataType

Class	**ApplicationError**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	This is a user-defined error that is associated with an element of an AUTOSAR interface. It is specific for the particular functionality or service provided by the AUTOSAR software component.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
errorCode	Integer	0..1	attr	The RTE generator is forced to assign this value to the corresponding error symbol. Note that for error codes certain ranges are predefined (see RTE specification).

Table A.10: ApplicationError

Class	**ApplicationPrimitiveDataType**
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes

▽

△

Class	ApplicationPrimitiveDataType			
Note	A primitive data type defines a set of allowed values. **Tags:**atp.recommendedPackage=ApplicationDataTypes			
Base	*ARElement*, *ARObject*, *ApplicationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.11: ApplicationPrimitiveDataType

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*, *Atp Blueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
element (ordered)	ApplicationRecord Element	*	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 ApplicationrecordData Type. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=preCompileTime

Table A.12: 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*			
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 ApplicationRecord Element 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.13: ApplicationRecordElement

Class	ArgumentDataPrototype
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface

▽

△

Class	**ArgumentDataPrototype**			
Note	An argument of an operation, much like a data element, but also carries direction information and is owned by a particular ClientServerOperation.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
direction	ArgumentDirection Enum	0..1	attr	This attribute specifies the direction of the argument prototype.
serverArgument ImplPolicy	ServerArgumentImpl PolicyEnum	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.14: 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.
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.15: ArgumentDirectionEnum

Class	***AutosarDataType*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::Datatypes			
Note	Abstract base class for user defined AUTOSAR data types for software.			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractImplementationDataType*, *ApplicationDataType*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
swDataDef Props	SwDataDefProps	0..1	aggr	The properties of this AutosarDataType.

Table A.16: AutosarDataType

Class	***BaseType*** (abstract)
Package	M2::MSR::AsamHdo::BaseTypes
Note	This abstract meta-class represents the ability to specify a platform dependent base type.

▽

△

Class	**BaseType** (abstract)			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	SwBaseType			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
baseType Definition	BaseTypeDefinition	1	aggr	This is the actual definition of the base type. **Tags:** xml.roleElement=false xml.roleWrapperElement=false xml.sequenceOffset=20 xml.typeElement=false xml.typeWrapperElement=false

Table A.17: BaseType

Class	**BaseTypeDirectDefinition**			
Package	M2::MSR::AsamHdo::BaseTypes			
Note	This BaseType is defined directly (as opposite to a derived BaseType)			
Base	*ARObject*, *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

▽

△

Class	**BaseTypeDirectDefinition**			
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.18: 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.
Literal	*Description*
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.19: 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*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
argument (ordered)	ArgumentDataPrototype	*	aggr	An argument of this ClientServerOperation **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=blueprintDerivationTime

▽

△

Class	**ClientServerOperation**			
fireAndForget	Boolean	0..1	attr	This attribute defines whether this method is a fire&forget method (true) or not (false). **Tags:**atp.Status=draft
possibleApError	ApApplicationError	*	ref	This reference identifies AdaptivePlatformApplication Errors as a possible error raised by the enclosing Client ServerOperation. **Tags:**atp.Status=draft
possibleApError Set	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 Ap ApplicationErrors separately. **Tags:**atp.Status=draft

Table A.20: ClientServerOperation

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=draft atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *ComGrant*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
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=draft
service Deployment	ServiceEvent Deployment	1	ref	This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies. **Tags:**atp.Status=draft

Table A.21: 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=draft atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *ComGrant*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
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=draft

▽

△

Class	ComFieldGrant			
role	FieldAccessEnum	1	attr	This attribute provides the ability to further specify the access to the ServiceInterface.field. **Tags:**atp.Status=draft
service Deployment	ServiceField Deployment	1	ref	This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies. **Tags:**atp.Status=draft

Table A.22: ComFieldGrant

Class	ComFindServiceGrant			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to grant the finding a service. **Tags:** atp.Status=draft atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
design	ComFindServiceGrant Design	0..1	ref	This reference identifies the ComFindServiceGrantDesign that the enclosing ComFindServiceGrant was created from. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
serviceInstance	AdaptivePlatform ServiceInstance	0..1	ref	This reference identifies the AdaptivePlatformService Instances for which the grant applies. **Tags:**atp.Status=draft

Table A.23: ComFindServiceGrant

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=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	ComEventGrant, ComFieldGrant, ComMethodGrant			
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=draft
serviceInstance	AdaptivePlatform ServiceInstance	1	ref	This reference identifies the applicable AdaptivePlatform ServiceInstance for which the grant applies. **Tags:**atp.Status=draft

Table A.24: 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=draft atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *ComGrant*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
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=draft
service Deployment	ServiceMethod Deployment	1	ref	This reference identifies the applicable deployment within the context of an AdaptivePlatformServiceInstance for which the grant applies. **Tags:**atp.Status=draft

Table A.25: 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=draft atp.recommendedPackage=Grants			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
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=draft
serviceInstance	AdaptivePlatform ServiceInstance	1	ref	This reference identifies the AdaptivePlatformService Instances for which the grant applies. **Tags:**atp.Status=draft

Table A.26: ComOfferServiceGrant

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 **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	CustomCppImplementationDataType, StdCppImplementationDataType			
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:** atp.Status=draft vh.latestBindingTime=preCompileTime
headerFile	String	0..1	attr	Configuration of the Header File with the custom class declaration. **Tags:**atp.Status=draft
namespace (ordered)	SymbolProps	*	aggr	This aggregation allows for the definition an own namespace for the enclosing CppImplementationData Type. **Tags:**atp.Status=draft
subElement (ordered)	CppImplementation DataTypeElement	*	aggr	This represents the collection of sub-elements of the enclosing CppImplementationDataType **Tags:**atp.Status=draft
template Argument (ordered)	CppTemplateArgument	*	aggr	This aggregation allows for the specification of properties of template arguments **Tags:**atp.Status=draft
typeEmitter	NameToken	0..1	attr	This attribute can be taken to control how the respective CppImplementationDataType is contributed to the language binding. **Tags:**atp.Status=draft
typeReference	CppImplementation DataType	0..1	ref	This reference shall be defined to define a type reference (a.k.a. typedef). **Tags:**atp.Status=draft

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. **Tags:**atp.Status=draft			
Base	*ARObject*, *AbstractImplementationDataTypeElement*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
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. **Tags:**atp.Status=draft

▽

△

Class	CppImplementationDataTypeElement			
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. **Tags:**atp.Status=draft

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. **Tags:**atp.Status=draft			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
allocator	Allocator	0..1	ref	This reference identifies the applicable allocator. **Tags:**atp.Status=draft
category	CategoryString	0..1	attr	This attribute shall be used to contribute further clarification regarding the semantics of the enclosing Cpp TemplateArgument. **Tags:**atp.Status=draft
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). **Tags:**atp.Status=draft
templateType	CppImplementation DataType	0..1	ref	This reference identifies the data type of the specific template argument required for the language binding. **Tags:**atp.Status=draft

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*			
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.

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*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
applicationDataType	ApplicationDataType	0..1	ref	This is the corresponding ApplicationDataType
implementationDataType	AbstractImplementationDataType	0..1	ref	This is the corresponding AbstractImplementationDataType.

Table A.31: DataTypeMap

Class	DdsEventDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	DDS configuration settings for an Event. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceEventDeployment*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
eventTopicAccessRule	DdsTopicAccessRule	0..1	ref	DDS Security access rule applicable to the DDS Topics used for the service interface event. **Tags:**atp.Status=draft
topicName	String	0..1	attr	Name of the DDS Topic associated with the Event. **Tags:**atp.Status=draft
transportProtocol	String	*	attr	This attribute defines over which Transport Layer Protocol(s) this event is intended to be sent. **Tags:**atp.Status=draft

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. **Tags:**atp.Status=draft			
Base	*ARObject*, *DdsQosProps*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
event	ServiceEventDeployment	1	ref	Reference to an event that is provided. **Tags:**atp.Status=draft

Table A.33: DdsEventQosProps

Class	DdsFieldDeployment
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment
Note	DDS configuration settings for a Field. **Tags:**atp.Status=draft
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceFieldDeployment*

▽

△

Class	DdsFieldDeployment			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
notifier	DdsEventDeployment	0..1	aggr	This aggregation represents the settings of the notifier. **Tags:**atp.Status=draft

Table A.34: DdsFieldDeployment

Class	DdsFieldQosProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment			
Note	Configuration properties of the Field interaction when using DDS as the underlying network binding. **Tags:**atp.Status=draft			
Base	*ARObject*, *DdsQosProps*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
field	ServiceField Deployment	1	ref	Reference to the field. **Tags:**atp.Status=draft

Table A.35: 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.Status=draft atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *DdsQosProps*, *DdsServiceInstanceProps*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *ProvidedApServiceInstance*, *Referrable*, *UploadablePackageElement*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
discoveryType	DdsServiceInstance DiscoveryTypeEnum	0..1	attr	Discovery protocol. **Tags:**atp.Status=draft
eventQosProps	DdsEventQosProps	*	aggr	List of configuration properties for the Events that are provided by the Service Instance. **Tags:**atp.Status=draft
fieldNotifierQosProps	DdsFieldQosProps	*	aggr	List of configuration properties for Field notifiers that are provided by the Service Instance. **Tags:**atp.Status=draft
resourceIdentifierType	DdsServiceInstance ResourceIdentifierType Enum	0..1	attr	Type of resource identification scheme. **Tags:**atp.Status=draft
serviceInstanceId	PositiveInteger	0..1	attr	Identification number that is used by DDS to identify DomainParticipants associated with an instance of the service. **Tags:**atp.Status=draft

Table A.36: 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. **Tags:**atp.Status=draft			
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. **Tags:**atp.Status=draft

Table A.37: 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.Status=draft atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *DdsQosProps*, *DdsServiceInstanceProps*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *RequiredApServiceInstance*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
blacklisted Version	DdsServiceVersion	*	aggr	Collection of blacklisted versions. **Tags:**atp.Status=draft
discoveryType	DdsServiceInstance DiscoveryTypeEnum	0..1	attr	Discovery protocol. **Tags:**atp.Status=draft
eventQosProps	DdsEventQosProps	*	aggr	List of configuration properties for the Events that are required by the Service Instance. **Tags:**atp.Status=draft
fieldNotifierQos Props	DdsFieldQosProps	*	aggr	List of configuration properties for Field notifiers that are required by the Service Instance. **Tags:**atp.Status=draft
requiredService InstanceId	AnyServiceInstanceId	0..1	attr	This attribute represents the ability to describe the required service instance ID. **Tags:**atp.Status=draft

Table A.38: 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. **Tags:**atp.Status=draft
Base	*ARObject*, *DdsQosProps*
Subclasses	DdsProvidedServiceInstance, DdsRequiredServiceInstance

▽

△

Class	***DdsServiceInstanceProps*** (abstract)			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
domainId	Integer	1	attr	This attribute identifies the DDS Domain the Service Instance shall join. **Tags:**atp.Status=draft

Table A.39: DdsServiceInstanceProps

Class	**DdsServiceInterfaceDeployment**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	DDS configuration settings for a ServiceInterface. **Tags:** atp.Status=draft atp.recommendedPackage=ServiceInterfaceDeployments			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *ServiceInterfaceDeployment*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
fieldReplyTopic Name	String	0..1	attr	Name of the DDS Reply Topic associated with the Field. **Tags:**atp.Status=draft
fieldRequest TopicName	String	0..1	attr	Name of the DDS Request Topic associated with the Field. **Tags:**atp.Status=draft
fieldTopics AccessRule	DdsTopicAccessRule	0..1	ref	DDS Security access rule applicable to the DDS Topics used for service interface field access methods (Get, Set). **Tags:**atp.Status=draft
methodReply TopicName	String	0..1	attr	Name of the DDS Reply Topic associated with the Method. **Tags:**atp.Status=draft
methodRequest TopicName	String	0..1	attr	Name of the DDS Request Topic associated with the Method. **Tags:**atp.Status=draft
methodTopics AccessRule	DdsTopicAccessRule	0..1	ref	DDS Security access rule applicable to the DDS Topics used for service interface methods. **Tags:**atp.Status=draft
serviceInterface Id	String	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. **Tags:**atp.Status=draft
transport Protocol	String	*	attr	This attribute defines over which Transport Layer Protocol(s) this Method is intended to be sent. **Tags:**atp.Status=draft

Table A.40: DdsServiceInterfaceDeployment

Class	**E2EProfileConfiguration**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::E2E

▽

△

Class	**E2EProfileConfiguration**			
Note	This element holds E2E profile specific configuration settings. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
clearFromValid ToInvalid	Boolean	0..1	attr	Clear monitoring window on transition from state Valid to state Invalid. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
e2eProfile Compatibility Props	E2EProfileCompatibility Props	0..1	ref	Reference to additional settings for the E2E state machine. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
profileName	NameToken	1	attr	Definition of the E2E profile. **Tags:**atp.Status=draft
windowSizeInit	PositiveInteger	0..1	attr	Size of the monitoring window of state Init for the E2E state machine. **Tags:**atp.Status=draft

▽

△

Class	**E2EProfileConfiguration**			
windowSize Invalid	PositiveInteger	0..1	attr	Size of the monitoring window of state Invalid for the E2E state machine. **Tags:**atp.Status=draft
windowSize Valid	PositiveInteger	0..1	attr	Size of the monitoring window of state Valid for the E2E state machine. **Tags:**atp.Status=draft

Table A.41: 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. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
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. **Tags:**atp.Status=draft
dataLength	PositiveInteger	0..1	attr	Length of payload including E2E header in bits. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
e2eProfile Configuration	E2EProfileConfiguration	0..1	ref	Reference to E2E profile configuration settings that are valid to protect the referenced event or field notifier. **Tags:**atp.Status=draft
event	ServiceEvent Deployment	0..1	ref	Reference to an event that is protected by the E2E profile. **Tags:**atp.Status=draft
maxDataLength	PositiveInteger	0..1	attr	Maximum length of payload including E2E header in bits. **Tags:**atp.Status=draft
minDataLength	PositiveInteger	0..1	attr	Minimum length of payload including E2E header in bits. **Tags:**atp.Status=draft

Table A.42: End2EndEventProtectionProps

Class	**End2EndMethodProtectionProps**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::E2E

▽

△

Class	**End2EndMethodProtectionProps**			
Note	This element allows to protect a method, a field setter or a field getter with an E2E profile. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
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. **Tags:**atp.Status=draft
dataLength	PositiveInteger	0..1	attr	Length of payload including E2E header in bits. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
maxDataLength	PositiveInteger	0..1	attr	Maximum length of payload including E2E header in bits. **Tags:**atp.Status=draft
method	ServiceMethod Deployment	0..1	ref	Reference to a method, a field getter or a field setter that is protected by the E2E profile. **Tags:**atp.Status=draft
minDataLength	PositiveInteger	0..1	attr	Minimum length of payload including E2E header in bits. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft

Table A.43: 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*

▽

△

Class	**EndToEndTransformationComSpecProps**			
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.44: EndToEndTransformationComSpecProps

Class	**EndToEndTransformationDescription**
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer

▽

△

Class	**EndToEndTransformationDescription**			
Note	EndToEndTransformationDescription holds these attributes which are profile specific and have the same value for all E2E transformers.			
Base	*ARObject*, *Describable*, *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	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.
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.

▽

▽

△

Class	EndToEndTransformationDescription			
				△ 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.45: EndToEndTransformationDescription

Class	EthernetCommunicationConnector			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Ethernet specific attributes to the CommunicationConnector.			
Base	*ARObject*, *CommunicationConnector*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
apApplication Endpoint	ApApplicationEndpoint	*	aggr	Collection of Application Addresses that are used on the CommunicationConnector. **Tags:**atp.Status=draft
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.
pncFilterData Mask	PositiveUnlimitedInteger	0..1	attr	Bit mask for Ethernet Payload used to configure the NM filter mask for the Network Management. **Tags:**atp.Status=obsolete
unicastNetwork Endpoint	NetworkEndpoint	0..1	ref	Network Endpoint that defines the IPAddress of the machine. **Tags:**atp.Status=draft

Table A.46: EthernetCommunicationConnector

Class	EthernetRawDataStreamClientMapping
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping

▽

△

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

Table A.47: EthernetRawDataStreamClientMapping

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

Table A.48: EthernetRawDataStreamLocalEndpointConfig

Class	***EthernetRawDataStreamMapping*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This meta-class serves as the abstract bases class for the ability to map a PortPrototype to a Ethernet-based communication channel. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *RawDataStreamMapping*, *Referrable*, *UploadablePackageElement*			
Subclasses	EthernetRawDataStreamClientMapping, EthernetRawDataStreamServerMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
localEndpoint Config	EthernetRawData StreamLocalEndpoint Config	0..1	aggr	This aggregation is used to configure the credentials of the endpoint. **Tags:**atp.Status=draft

▽

△

Class	***EthernetRawDataStreamMapping*** (abstract)			
socketOption	String	*	attr	This attribute represents the ability to specify non-formal socket options that might only be valid for specific platforms. AUTOSAR does not define a standardized meaning for the possible values of this attribute. **Tags:**atp.Status=draft
tlsSecureCom Props	TlsSecureComProps	0..1	ref	This reference provides the ability to define TLS-related properties for the enclosing SocketRawDataStream Mapping. **Tags:**atp.Status=draft

Table A.49: EthernetRawDataStreamMapping

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

Table A.50: EthernetRawDataStreamRemoteClientConfig

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

Table A.51: EthernetRawDataStreamRemoteServerConfig

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

Table A.52: EthernetRawDataStreamServerMapping

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. **Tags:**atp.Status=draft			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
hasGetter	Boolean	1	attr	This attribute controls whether read access is foreseen to this field. **Tags:**atp.Status=draft
hasNotifier	Boolean	1	attr	This attribute controls whether a notification semantics is foreseen to this field. **Tags:**atp.Status=draft
hasSetter	Boolean	1	attr	This attribute controls whether write access is foreseen to this field. **Tags:**atp.Status=draft

Table A.53: Field

Enumeration	**FieldAccessEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::GrantDesign::ComGrant
Note	This meta-class provides values that qualify access to a field. **Tags:**atp.Status=draft
Literal	***Description***
getter	Access to the getter of the Field. **Tags:** atp.EnumerationLiteralIndex=0 atp.Status=draft
getterSetter	Access to getter and setter of the field **Tags:** atp.EnumerationLiteralIndex=2 atp.Status=draft

▽

△

Enumeration	**FieldAccessEnum**
setter	Access to the setter of the Field. **Tags:** atp.EnumerationLiteralIndex=1 atp.Status=draft

Table A.54: FieldAccessEnum

Class	**FieldSenderComSpec**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ComSpec			
Note	Port specific communication attributes for a Field that is defined in a ServiceInterface. **Tags:**atp.Status=draft			
Base	*ARObject*, *PPortComSpec*, *SenderComSpec*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
initValue	ValueSpecification	1	aggr	Initial value for a Field that is set before the Service Interface is offered. **Tags:**atp.Status=draft

Table A.55: FieldSenderComSpec

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*			
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.56: 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=draft atp.recommendedPackage=IamRemoteSubjects			
Base	*ARElement*, *ARObject*, *AbstractIamRemoteSubject*, *CollectableElement*, *Identifiable*, *Multilanguage Referrable*, *PackageableElement*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
localIpSecRule	IPSecRule	*	ref	This reference is used to describe theRemoteSubjects local IPSecRules. **Tags:**atp.Status=draft

Table A.57: 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*			
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.58: IPSecRule

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	*ARObject*, *CollectableElement*, *FibexElement*, *IPdu*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Pdu*, *Referrable*			
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:** atpVariation **Tags:**vh.latestBindingTime=postBuild
iSignalToPdu Mapping	ISignalToIPduMapping	*	aggr	Definition of SignalToIPduMappings included in the Signal IPdu. atpVariation: The content of a PDU can be variable. **Stereotypes:** atpVariation **Tags:**vh.latestBindingTime=postBuild
unusedBit Pattern	Integer	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.59: ISignalIPdu

Class	**ISignalToIPduMapping**
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication

▽

△

Class	**ISignalToIPduMapping**			
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*			
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.
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 ▽

▽

△

Class	ISignalToIPduMapping			
				△ 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.60: 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*			
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.
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.61: ISignalTriggering

Class	IamModuleInstantiation			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement			
Note	This meta-class represents the ability to define a definition of an IAM instantiation. **Tags:**atp.Status=draft			
Base	*ARObject*, *AdaptiveModuleInstantiation*, *Identifiable*, *MultilanguageReferrable*, *NonOsModule Instantiation*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
grant	Grant	*	ref	This reference identifies the applicable Grants for this Iam ModuleInstantiation. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=grant atp.Status=draft
localCom AccessControl Enabled	Boolean	0..1	attr	This switch activates the policy enforcement in Communication Management on local applications. **Tags:**atp.Status=draft

▽

△

Class	IamModuleInstantiation			
remoteAccess ControlEnabled	Boolean	0..1	attr	This switch activates the check of the remote subject. **Tags:**atp.Status=draft

Table A.62: IamModuleInstantiation

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*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, *Abstract SignalBasedToISignalTriggeringMapping*, *AdaptiveModuleInstantiation*, AdaptiveSwcInternalBehavior, ApApplicationEndpoint, ApplicationEndpoint, ApplicationError, ArtifactChecksum, *AtpBlueprint*, *Atp Blueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BuildActionEntity*, BuildActionEnvironment, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *Comm ConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, CryptoCertificate, CryptoKeySlot, CryptoProvider, *CryptoServiceMapping*, DataPrototypeGroup, DataTransformation, DdsDomainRange, DependencyOnArtifact, DeterministicClientResourceNeeds, *DiagEventDebounceAlgorithm*, Diagnostic ConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticFunction InhibitSource, *DiagnosticRoutineSubfunction*, DltApplication, DltArgument, DltMessage, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, E2EProfileConfiguration, End2EndEventProtectionProps, End2EndMethodProtectionProps, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, Event Handler, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeature MapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeature Restriction, FMFeatureSelection, FieldMapping, FireAndForgetMapping, FlexrayArTpNode, FlexrayTp PduPool, *FrameTriggering*, GeneralParameter, GlobalSupervision, GlobalTimeGateway, *GlobalTime Master*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *Ident Caption*, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacMulticastGroup, McDataInstance, MemorySection, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRouting Group, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PersistencyDeploymentElement*, *PersistencyInterfaceElement*, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlot Mapping, RecoveryNotification, ResourceConsumption, ResourceGroup, RootSwClusterDesign ComponentPrototype, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, Rpt Container, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptService Point, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunication FreshnessProps, SecurityEventContextProps, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInterfaceElementSecureComConfig, *ServiceMethodDeployment*, *ServiceNeeds*, SignalService TranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SupervisionMode, SupervisionMode Condition, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SystemMapping, SystemMemoryUsage, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, Tls JobMapping, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, *TransformationProps*, TransformationTechnology, Trigger, UcmDescription, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***Identifiable*** (abstract)			
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.63: Identifiable

Class	***ImplementationProps*** (abstract)
Package	M2::AUTOSARTemplates::CommonStructure::Implementation
Note	Defines a symbol to be used as (depending on the concrete case) either a complete replacement or a prefix when generating code artifacts.

▽

△

Class	**ImplementationProps** (abstract)			
Base	*ARObject*, *Referrable*			
Subclasses	BswSchedulerNamePrefix, ExecutableEntityActivationReason, SectionNamePrefix, SymbolProps, SymbolicNameProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
symbol	CIdentifier	0..1	attr	The symbol to be used as (depending on the concrete case) either a complete replacement or a prefix.

Table A.64: ImplementationProps

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*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
initialDelayMaxValue	TimeValue	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	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=draft atp.recommendedPackage=IamRemoteSubjects			
Base	*ARElement*, *ARObject*, *AbstractIamRemoteSubject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
authenticConnectionProps	IplamAuthenticConnectionProps	*	aggr	Definition of IP rules assigned to the IplamRemoteSubject. **Tags:**atp.Status=draft

Table A.66: IplamRemoteSubject

Class	**Ipv4Configuration**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::EthernetTopology			
Note	Internet Protocol version 4 (IPv4) configuration.			
Base	*ARObject*, *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*			
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.

▽

△

Class	Ipv6Configuration			
ipv6Address Source	Ipv6AddressSource Enum	0..1	attr	Defines how the node obtains its IP address.

Table A.68: Ipv6Configuration

Class	Machine			
Package	M2::AUTOSARTemplates::AdaptivePlatform::MachineManifest			
Note	Machine that represents an Adaptive Autosar Software Stack. **Tags:** atp.Status=draft atp.recommendedPackage=Machines			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
default Application Timeout	EnterExitTimeout	0..1	aggr	This aggration defines a default timeout in the context of a given Machine with respect to the launching and termination of applications. **Tags:**atp.Status=draft
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, environment Variable.variationPoint.shortLabel atp.Status=draft
machineDesign	MachineDesign	1	ref	Reference to the MachineDesign this Machine is implementing. **Tags:**atp.Status=draft
module Instantiation	AdaptiveModule Instantiation	*	aggr	Configuration of Adaptive Autosar module instances that are running on the machine. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=moduleInstantiation.shortName atp.Status=draft
processor	Processor	1..*	aggr	This represents the collection of processors owned by the enclosing machine. **Tags:**atp.Status=draft
secure Communication Deployment	SecureCommunication Deployment	*	aggr	Deployment of secure communication protocol configuration settings to crypto module entities. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=secureCommunicationDeployment.short Name atp.Status=draft
trustedPlatform Executable LaunchBehavior	TrustedPlatform ExecutableLaunch BehaviorEnum	1	attr	This attribute controls the behavior of how authentication affects the ability to launch for each Executable. **Tags:**atp.Status=draft

Table A.69: 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*			
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	1..*	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.70: NetworkEndpoint

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*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
packingByte Order	ByteOrderEnum	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	1	ref	Reference to a I-Pdu, N-Pdu or NmPdu that is transmitted in the Frame.
startPosition	Integer	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. ▽

▽

△

Class	<<atpPrototype>> **PduToFrameMapping**			
				△ 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.71: 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*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
iPdu	Pdu	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:** atpVariation **Tags:**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.72: 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*, LogAndTraceInterface, ModeSwitchInterface, *PersistencyInterface*, *PlatformHealthManagementInterface*, SecurityEventReportInterface, Service Interface, TriggerInterface			
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 atp.Status=draft

Table A.73: PortInterface

Class	**Process**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class provides information required to execute the referenced executable. **Tags:** atp.Status=draft atp.recommendedPackage=Processes			
Base	*ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
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. **Tags:**atp.Status=draft
deterministic Client	DeterministicClient	0..1	ref	This reference adds further execution characteristics for deterministic clients. **Tags:**atp.Status=draft
executable	Executable	0..1	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
functionCluster Affiliation	String	0..1	attr	This attribute specifies which functional cluster the process is affiliated with. **Tags:**atp.Status=draft
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 **Tags:**atp.Status=draft
preMapping	Boolean	0..1	attr	This attribute describes whether the executable is preloaded into the memory. **Tags:**atp.Status=draft

▽

△

Class	Process			
processState Machine	ModeDeclarationGroup Prototype	0..1	aggr	Set of Process States that are defined for the process. **Tags:**atp.Status=draft
securityEvent	SecurityEventDefinition	*	ref	The reference identifies the collection of SecurityEvents that can be reported by the enclosing SoftwareCluster. **Stereotypes:** atpSplitable; atpUriDef **Tags:** atp.Splitkey=securityEvent atp.Status=draft
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations. **Tags:**atp.Status=draft

Table A.74: 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. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Subclasses	DdsProvidedServiceInstance, ProvidedSomeipServiceInstance, ProvidedUserDefinedServiceInstance			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.75: 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.Status=draft atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *ProvidedApServiceInstance*, *Referrable*, *Uploadable PackageElement*			
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. **Tags:**atp.Status=draft
eventProps	SomeipEventProps	*	aggr	Configuration settings for individual events that are provided by the ServiceInstance. **Tags:**atp.Status=draft

▽

△

Class	**ProvidedSomeipServiceInstance**			
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
method ResponseProps	SomeipMethodProps	*	aggr	Configuration settings for individual methods that are provided by the ServiceInstance. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
providedEvent Group	SomeipProvidedEvent Group	*	aggr	List of EventGroups that are provided by the Service Instance. **Tags:**atp.Status=draft
sdServerConfig	SomeipSdServer ServiceInstanceConfig	1	ref	Server specific configuration settings relevant for the SOME/IP service discovery. **Tags:**atp.Status=draft
serviceInstance Id	PositiveInteger	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. **Tags:**atp.Status=draft

Table A.76: 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.Status=draft atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *ProvidedApServiceInstance*, *Referrable*, *Uploadable PackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.77: ProvidedUserDefinedServiceInstance

Class	**RawDataStreamClientInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This meta-class represents the necessary capabilities for raw data streaming on the client side, i.e. the streaming of data that do not undergo any serialization. Each RawDataStreamClientInterface supports the following capabilities without further modeling: • connect: set up the communication channel • shutdown: close the communication channel • write: send data down the communication channel • read: access incoming data on the communication channel **Tags:** atp.Status=draft atp.recommendedPackage=RawDataStreamInterfaces			
Base	*ARElement*, *ARObject*, *AbstractRawDataStreamInterface*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.78: RawDataStreamClientInterface

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

Table A.79: RawDataStreamEthernetTcpUdpCredentials

Class	**RawDataStreamEthernetUdpCredentials**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::RawDataStreamMapping			
Note	This-meta-class represents the ability to create a configuration of network credentials for a raw data stream connection over UDP. **Tags:**atp.Status=draft			
Base	*ARObject*, *AbstractRawDataStreamEthernetCredentials*, *Describable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.80: RawDataStreamEthernetUdpCredentials

Class	***RawDataStreamGrant*** (abstract)
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::IdentityAccessManagement

▽

△

Class	**RawDataStreamGrant** (abstract)			
Note	This abstract meta-class represents the ability to define the IAM configuration for a RawDataStream on deployment level. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Grant*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	EthernetRawDataStreamGrant			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
design	RawDataStreamGrant Design	0..1	ref	This reference identifies the RawDataStreamGrantDesign that the enclosing RawDataStreamEventGrant was created from. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft

Table A.81: RawDataStreamGrant

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

Table A.82: RawDataStreamMapping

Class	**RawDataStreamServerInterface**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface

▽

△

Class	**RawDataStreamServerInterface**			
Note	This meta-class represents the necessary capabilities for raw data streaming on the server side, i.e. the streaming of data that do not undergo any serialization. Each RawDataStreamServerInterface supports the following capabilities without further modeling: • waitForConnection: wait until a communication channel is set up. • shutdown: close the communication channel • write: send data down the communication channel • read: access incoming data on the communication channel **Tags:** atp.Status=draft atp.recommendedPackage=RawDataStreamInterfaces			
Base	*ARElement*, *ARObject*, *AbstractRawDataStreamInterface*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.83: RawDataStreamServerInterface

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.84: Referrable

Class	**RequestResponseDelay**			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances			
Note	Time to wait before answering the query.			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
maxValue	TimeValue	1	attr	Maximum allowable response delay to entries received by multicast in seconds.

▽

△

Class	RequestResponseDelay			
minValue	TimeValue	1	attr	Minimum allowable response delay to entries received by multicast in seconds.

Table A.85: 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. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Subclasses	DdsRequiredServiceInstance, RequiredSomeipServiceInstance, RequiredUserDefinedServiceInstance			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.86: 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.Status=draft atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *RequiredApServiceInstance*, *Uploadable PackageElement*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
blacklisted Version	SomeipServiceVersion	*	aggr	Collection of blacklisted versions. **Tags:**atp.Status=draft
capability Record (ordered)	TagWithOptionalValue	*	aggr	A sequence of records to store arbitrary name/value pairs conveying additional information about the named service. **Tags:**atp.Status=draft
methodRequest Props	SomeipMethodProps	*	aggr	Configuration settings for individual methods that are requested by the ServiceInstance. **Tags:**atp.Status=draft
requiredEvent Group	SomeipRequiredEvent Group	*	aggr	List of EventGroups that are used by the RequiredService Instance. **Tags:**atp.Status=draft
requiredMinor Version	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. **Tags:**atp.Status=draft
requiredService InstanceId	AnyServiceInstanceId	0..1	attr	This attribute represents the ability to describe the required service instance ID. **Tags:**atp.Status=draft

▽

△

Class	**RequiredSomeipServiceInstance**			
sdClientConfig	SomeipSdClientService InstanceConfig	1	ref	Client specific configuration settings relevant for the SOME/IP service discovery. **Tags:**atp.Status=draft
versionDriven FindBehavior	ServiceVersion AcceptanceKindEnum	0..1	attr	Defines the service discovery find behavior. **Tags:**atp.Status=draft

Table A.87: 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.Status=draft atp.recommendedPackage=ServiceInstances			
Base	*ARElement*, *ARObject*, *AdaptivePlatformServiceInstance*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *RequiredApServiceInstance*, *Uploadable PackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.88: RequiredUserDefinedServiceInstance

Enumeration	**SOMEIPTransformerSessionHandlingEnum**
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer
Note	Enables or disable session handling for SOME/IP transformer
Literal	***Description***
sessionHandling Active	The SOME/IP Transformer shall use session handling **Tags:**atp.EnumerationLiteralIndex=0
sessionHandling Inactive	The SOME/IP Transformer doesn't use session handling **Tags:**atp.EnumerationLiteralIndex=1

Table A.89: SOMEIPTransformerSessionHandlingEnum

Class	**SecOcSecureComProps**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	Configuration of AUTOSAR SecOC. **Tags:** atp.Status=draft atp.recommendedPackage=SecureComProps			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *SecureComProps*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
authAlgorithm	String	0..1	attr	This attribute defines the authentication algorithm used for MAC generation and verification. **Tags:**atp.Status=draft

▽

△

Class	SecOcSecureComProps			
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. **Tags:**atp.Status=draft
freshnessValue Length	PositiveInteger	0..1	attr	This attribute defines the complete length in bits of the Freshness Value. **Tags:**atp.Status=draft
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. In other words this attribute defines the length of the authenticated Message. **Tags:**atp.Status=draft
jobRequirement	SecOcJobRequirement	*	aggr	Collection of cryptographic job requirements. **Tags:**atp.Status=draft

Table A.90: SecOcSecureComProps

Class	*SecureComProps* (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	This meta-class defines a communication security protocol and its configuration settings. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	DdsSecureComProps, SecOcSecureComProps, TlsSecureComProps			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.91: SecureComProps

Class	SecureCommunicationAuthenticationProps			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Authentication properties used to configure SecuredIPdus.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
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.92: SecureCommunicationAuthenticationProps

Class	SecureCommunicationFreshnessProps			
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::FibexCore::CoreCommunication			
Note	Freshness properties used to configure SecuredIPdus.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**SecureCommunicationFreshnessProps**			
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.
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.93: 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*			
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	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	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.

▽

△

Class	**SecureCommunicationProps**			
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.94: 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	*ARObject*, *CollectableElement*, *FibexElement*, *IPdu*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Pdu*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
authentication Props	SecureCommunication AuthenticationProps	0..1	ref	Reference to authentication properties that are valid for this SecuredIPdu.
freshnessProps	SecureCommunication FreshnessProps	0..1	ref	Reference to freshness properties that are valid for this SecuredIPdu.
payload	PduTriggering	1	ref	Reference to a Pdu that will be protected against unauthorized manipulation and replay attacks.
secure Communication Props	SecureCommunication Props	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.95: SecuredIPdu

Enumeration	**SerializationTechnologyEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment
Note	This enumeration allows to choose a Serialization Technology. **Tags:**atp.Status=draft
Literal	***Description***
signalBased	Signal-Based serializer. **Tags:** atp.EnumerationLiteralIndex=1 atp.Status=draft
someip	SOME/IP Serializer **Tags:** atp.EnumerationLiteralIndex=0 atp.Status=draft

Table A.96: 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. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DdsEventDeployment, SomeipEventDeployment, UserDefinedEventDeployment			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
event	VariableDataPrototype	0..1	ref	Reference to an Event that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
trigger	Trigger	0..1	ref	Reference to a Trigger that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft

Table A.97: 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. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	DdsFieldDeployment, SomeipFieldDeployment, UserDefinedFieldDeployment			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
field	Field	1	ref	Reference to a Field that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft

Table A.98: 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. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Subclasses	DdsServiceInstanceToMachineMapping, SomeipServiceInstanceToMachineMapping, UserDefined ServiceInstanceToMachineMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
communication Connector	Communication Connector	0..1	ref	Reference to the Machine to which the ServiceInstance is mapped. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
serviceInstance	AdaptivePlatform ServiceInstance	*	ref	Reference to a ServiceInstance that is mapped to the Machine. **Tags:**atp.Status=draft

Table A.99: 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.Status=draft atp.recommendedPackage=ServiceInstanceToPortPrototypeMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadablePackageElement*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
portPrototype	PortPrototype	0..1	iref	Reference to a specific PortPrototype that represents the ServiceInstance. **Tags:**atp.Status=draft **InstanceRef implemented by:**PortPrototypeIn ExecutableInstanceRef

△

Class	ServiceInstanceToPortPrototypeMapping			
process	Process	0..1	ref	Reference to the Process in which the enclosing Service InstanceToPortPrototypeMapping is executed. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=process atp.Status=draft
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 **Tags:**atp.Status=draft
serviceInstance	AdaptivePlatform ServiceInstance	0..1	ref	Reference to a ServiceInstance that is represented in the Software Component by the mapped group of Port Prototypes. **Tags:**atp.Status=draft

Table A.100: 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=draft atp.recommendedPackage=ServiceInstanceToSignalMapping			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
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=draft
fieldMapping	SignalBasedFieldToI SignalTriggering Mapping	*	aggr	Mapping of a field to ISignalTriggerings. **Tags:**atp.Status=draft
methodMapping	SignalBasedMethodToI SignalTriggering Mapping	0..1	aggr	Mapping of a method to ISignalTriggerings. **Tags:**atp.Status=draft
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=draft
triggerMapping	SignalBasedTriggerToI SignalTriggering Mapping	*	aggr	Mapping of a trigger to an ISignalTriggering. **Tags:**atp.Status=draft

Table A.101: ServiceInstanceToSignalMapping

Class	ServiceInterface
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface

▽

△

Class	**ServiceInterface**			
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. **Tags:** atp.Status=draft atp.recommendedPackage=ServiceInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
event	VariableDataPrototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
field	Field	*	aggr	This represents the collection of fields defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=40
majorVersion	PositiveInteger	0..1	attr	Major version of the service contract. **Tags:** atp.Status=draft xml.sequenceOffset=10
method	ClientServerOperation	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=50
minorVersion	PositiveInteger	0..1	attr	Minor version of the service contract. **Tags:** atp.Status=draft xml.sequenceOffset=20
trigger	Trigger	*	aggr	This represents the collection of triggers defined in the context of a ServiceInterface. **Stereotypes:** atpVariation **Tags:** atp.Status=draft vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=60

Table A.102: 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. **Tags:**atp.Status=draft

▽

△

Class	**ServiceInterfaceDeployment** (abstract)			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadablePackageElement*			
Subclasses	DdsServiceInterfaceDeployment, SomeipServiceInterfaceDeployment, UserDefinedServiceInterfaceDeployment			
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. **Tags:**atp.Status=draft
fieldDeployment	ServiceField Deployment	*	aggr	Middleware transport layer specific configuration settings for a Field that is defined in the ServiceInterface. **Tags:**atp.Status=draft
method Deployment	ServiceMethod Deployment	*	aggr	Middleware transport layer specific configuration settings for a method that is defined in the ServiceInterface. **Tags:**atp.Status=draft
serviceInterface	ServiceInterface	0..1	ref	Reference to a ServiceInterface that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft

Table A.103: ServiceInterfaceDeployment

Class	**ServiceInterfaceElementSecureComConfig**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	This element allows to secure the communication of the referenced ServiceInterface element. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
dataId	PositiveInteger	0..1	attr	This attribute defines a unique numerical identifier for the referenced ServiceInterface element. **Tags:**atp.Status=draft
event	ServiceEvent Deployment	0..1	ref	Reference to an event that is protected by a security protocol. **Tags:**atp.Status=draft
fieldNotifier	ServiceField Deployment	0..1	ref	Reference to a field notifier that is protected by a security protocol. **Tags:**atp.Status=draft
freshnessValueId	PositiveInteger	0..1	attr	This attribute defines the Id of the Freshness Value. **Tags:**atp.Status=draft
getterCall	ServiceField Deployment	0..1	ref	Reference to a field getter call message that is protected by a security protocol. **Tags:**atp.Status=draft
getterReturn	ServiceField Deployment	0..1	ref	Reference to a field getter return message that is protected by a security protocol. **Tags:**atp.Status=draft
methodCall	ServiceMethod Deployment	0..1	ref	Reference to a method call message that is protected by a security protocol. **Tags:**atp.Status=draft

▽

△

Class	**ServiceInterfaceElementSecureComConfig**			
methodReturn	ServiceMethod Deployment	0..1	ref	Reference to a method return message that is protected by a security protocol. **Tags:**atp.Status=draft
setterCall	ServiceField Deployment	0..1	ref	Reference to a field setter call message that is protected by a security protocol. **Tags:**atp.Status=draft
setterReturn	ServiceField Deployment	0..1	ref	Reference to a field setter return message that is protected by a security protocol. **Tags:**atp.Status=draft

Table A.104: 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. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	SomeipMethodDeployment, UserDefinedMethodDeployment			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
method	ClientServerOperation	0..1	ref	Reference to a method that is deployed to a middleware transport layer. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft

Table A.105: ServiceMethodDeployment

Enumeration	**ServiceVersionAcceptanceKindEnum**
Package	M2::AUTOSARTemplates::SystemTemplate::Fibex::Fibex4Ethernet::ServiceInstances
Note	Defined the possible acceptance kinds for required service instances. **Tags:**atp.Status=draft
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 atp.Status=draft
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 atp.Status=draft

Table A.106: ServiceVersionAcceptanceKindEnum

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. **Tags:**atp.Status=draft			
Base	*ARObject*			
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft

Table A.107: 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.Status=draft atp.recommendedPackage=SomeipDataPrototypeTransformationPropss			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
someip Transformation Props	ApSomeip TransformationProps	0..1	ref	This reference represents the ability to define data transformation props specifically for a SOME/IP serialization. **Tags:**atp.Status=draft

Table A.108: SomeipDataPrototypeTransformationProps

Class	**SomeipEventDeployment**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment

▽

△

Class	**SomeipEventDeployment**			
Note	SOME/IP configuration settings for an Event. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceEventDeployment*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
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. **Tags:**atp.Status=draft
eventId	PositiveInteger	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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
separationTime	TimeValue	0..1	attr	Sets the duration of the minimum time in seconds SOME/IP shall wait between the transmissions of segments. **Tags:**atp.Status=draft
serializer	SerializationTechnology Enum	0..1	attr	Defines which serialization technology shall be used. **Tags:**atp.Status=draft
transport Protocol	TransportLayerProtocol Enum	1	attr	This attribute defines over which Transport Layer Protocol this event is intended to be sent. **Tags:**atp.Status=draft

Table A.109: SomeipEventDeployment

Class	**SomeipEventGroup**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	Grouping of events and notification events inside a ServiceInterface in order to allow subscriptions. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
event	SomeipEvent Deployment	*	ref	Reference to an event that is part of the EventGroup. **Tags:**atp.Status=draft
eventGroupId	PositiveInteger	1	attr	Unique Identifier that identifies the EventGroup in SOME/IP. This Identifier is sent as Eventgroup ID in SOME/IP Service Discovery messages. **Tags:**atp.Status=draft

Table A.110: 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. **Tags:**atp.Status=draft			
Base	*ARObject*			
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. **Tags:**atp.Status=draft
event	SomeipEvent Deployment	0..1	ref	Reference to the event for which the SomeipEventProps are applicable. **Tags:**atp.Status=draft

Table A.111: SomeipEventProps

Class	SomeipFieldDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a Field. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceFieldDeployment*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
get	SomeipMethod Deployment	0..1	aggr	This aggregation represents the setting of the get method. **Tags:**atp.Status=draft
notifier	SomeipEvent Deployment	0..1	aggr	This aggregation represents the settings of the notifier. **Tags:**atp.Status=draft
set	SomeipMethod Deployment	0..1	aggr	This aggregation represents the settings of the set method **Tags:**atp.Status=draft

Table A.112: SomeipFieldDeployment

Class	SomeipMethodDeployment			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a Method. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *ServiceMethodDeployment*			
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. **Tags:**atp.Status=draft

▽

△

Class	**SomeipMethodDeployment**			
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
methodId	PositiveInteger	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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
transport Protocol	TransportLayerProtocol Enum	1	attr	This attribute defines over which Transport Layer Protocol this method is intended to be sent. **Tags:**atp.Status=draft

Table A.113: 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. **Tags:**atp.Status=draft			
Base	*ARObject*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**SomeipMethodProps**			
collectionProps	SomeipCollectionProps	0..1	aggr	Collection of timing attributes configurable for a method that is provided or requested by a Service Instance. **Tags:**atp.Status=draft
method	SomeipMethod Deployment	0..1	ref	Reference to the method for which the SomeipMethod Props are applicable. **Tags:**atp.Status=draft

Table A.114: 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. **Tags:**atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
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. **Tags:**atp.Status=draft
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. **Tags:**atp.Status=draft
ipv4MulticastIp Address	Ip4AddressString	0..1	attr	Multicast IPv4 Address that is transmitted in the Event GroupSubscribeAck message. **Tags:**atp.Status=draft
ipv6MulticastIp Address	Ip6AddressString	0..1	attr	Multicast IPv6 Address that is transmitted in the Event GroupSubscribeAck message. **Tags:**atp.Status=draft
multicast Threshold	PositiveInteger	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. **Tags:**atp.Status=draft
sdServerEvent GroupTiming Config	SomeipSdServerEvent GroupTimingConfig	0..1	ref	Server Timing configuration settings that are EventGroup specific. **Tags:**atp.Status=draft

Table A.115: 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. **Tags:**atp.Status=draft			
Base	*ARObject*, *Referrable*			
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. **Tags:**atp.Status=draft
sdClientEvent GroupTiming Config	SomeipSdClientEvent GroupTimingConfig	1	ref	Client Timing configuration settings that are EventGroup specific. **Tags:**atp.Status=draft

Table A.116: 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*			
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	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.117: 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*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**SomeipSdClientServiceInstanceConfig**			
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.
serviceFind TimeToLive	PositiveInteger	1	attr	This attribute represents the ability to define the time in seconds the service find is valid.

Table A.118: 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*			
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).
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	1	attr	Defines the time in seconds the service offer is valid.

Table A.119: 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.Status=draft atp.recommendedPackage=ServiceInstanceToMachineMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *ServiceInstanceToMachineMapping*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	**SomeipServiceInstanceToMachineMapping**			
tcpPort	ApApplicationEndpoint	0..1	ref	TcpPort configuration that is used for Method and Event communication in IP-Unicast case. During SOME/IP Service Discovery: PortNumber that is sent in the SD-Offer Message to client (answer on SD-find) or clients (SD-offer). Method: This is the destination-port where the server accepts the method call messages (from the clients). This is the source-port where the server sends the method response messages (to the client). Event: This is the event source-port where the server sends the event messages to the subscribed clients in IP-Unicast case. **Tags:**atp.Status=draft
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 in case data collection is enabled. **Tags:**atp.Status=draft
udpPort	ApApplicationEndpoint	0..1	ref	UdpPort configuration that is used for Method and Event communication in IP-Unicast case. During SOME/IP Service Discovery: PortNumber that is sent in the SD-Offer Message to client (answer on SD-find) or clients (SD-offer). Method: This is the destination-port where the server accepts the method call messages (from the clients). This is the source-port where the server sends the method response messages (to the client). Event: This is the event source-port where the server sends the event messages to the subscribed clients in IP-Unicast case. **Tags:**atp.Status=draft

Table A.120: SomeipServiceInstanceToMachineMapping

Class	**SomeipServiceInterfaceDeployment**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	SOME/IP configuration settings for a ServiceInterface. **Tags:** atp.Status=draft atp.recommendedPackage=ServiceInterfaceDeployments			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *ServiceInterfaceDeployment*, *UploadablePackageElement*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
eventGroup	SomeipEventGroup	*	aggr	SOME/IP EventGroups that are defined within the SOME/IP ServiceClass. **Tags:**atp.Status=draft
serviceInterface Id	PositiveInteger	1	attr	Unique Identifier that identifies the ServiceInterface in SOME/IP. This Identifier is sent as Service ID in SOME/IP Service Discovery messages. **Tags:**atp.Status=draft

▽

△

Class	SomeipServiceInterfaceDeployment			
serviceInterface Version	SomeipServiceVersion	1	aggr	The SOME/IP major and minor Version of the Service. **Tags:**atp.Status=draft

Table A.121: 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. **Tags:**atp.Status=draft			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
majorVersion	PositiveInteger	0..1	attr	Major Version of the ServiceInterface. **Tags:** atp.Status=draft xml.sequenceOffset=10
minorVersion	PositiveInteger	1	attr	Minor Version of the ServiceInterface. **Tags:** atp.Status=draft xml.sequenceOffset=20

Table A.122: 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.Status=draft atp.recommendedPackage=CppImplementationDataTypes			
Base	*ARElement*, *ARObject*, *AbstractImplementationDataType*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *AutosarDataType*, *CollectableElement*, *CppImplementationDataType*, *CppImplementationDataTypeContextTarget*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.123: 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 Data

▽

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
	△ Types in the programming language (or in AUTOSAR). This is mainly expressed by properties like swRecordLayout and swCalprmAxisSet • Implementation aspects, mainly expressed by swImplPolicy, swVariableAccessImplPolicy, sw AddrMethod, swPointerTagetProps, baseType, implementationDataType and additionalNative TypeQualifier • Access policy for the MCD system, mainly expressed by swCalibrationAccess • Semantics of the data element, mainly expressed by compuMethod and/or unit, dataConstr, invalidValue • Code generation policy provided by swRecordLayout **Tags:**vh.latestBindingTime=codeGenerationTime			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
additionalNative TypeQualifier	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
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 ▽

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
				△ • 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
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

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
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
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

▽

△

Class	<<atpVariation>> **SwDataDefProps**			
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.124: 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*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
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

▽

△

Class	SwTextProps			
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.125: 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*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.126: SymbolProps

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=draft atp.recommendedPackage=IamRemoteSubjects			
Base	*ARElement*, *ARObject*, *AbstractIamRemoteSubject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
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=draft
acceptedRemoteCertificate	CryptoServiceCertificate	*	ref	This reference is used to identify a remote node by means of the certificate. **Tags:**atp.Status=draft
certCommonName	String	0..1	attr	This attribute defines the common name (CN) of the certificate of the remote peer. **Tags:**atp.Status=draft
derivedCertificateAccepted	Boolean	0..1	attr	This attribute defines whether a derivedCertificate is accepted (true) or not (false). **Tags:**atp.Status=draft
iamRelevantTlsSecureComProps	TlsSecureComProps	*	ref	This reference defines the local TlsSecureComProps that are relevant for IAM. **Tags:**atp.Status=draft

Table A.127: TlsIamRemoteSubject

Class	TlsSecureComProps			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::SecureCommunication			
Note	Configuration of the Transport Layer Security protocol (TLS). **Tags:** atp.Status=draft atp.recommendedPackage=SecureComProps			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *SecureComProps*			
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. **Tags:**atp.Status=draft
tlsCipherSuite	TlsCryptoCipherSuite	*	aggr	Collection of supported cipher suites that are used to negotiate the security settings for a network connection defined by the ServiceInstanceToMachineMapping. **Tags:**atp.Status=draft

Table A.128: TlsSecureComProps

Class	TlvDataIdDefinition			
Package	M2::AUTOSARTemplates::SystemTemplate::Transformer			
Note	This meta-class represents the ability to define the tlvDataId.			
Base	*ARObject*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
id	PositiveInteger	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 Implementation DataType Element	AbstractImplementation DataTypeElement	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.Status=draft atp.recommendedPackage=TransformationPropsToServiceInterfaceElementMappings			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	**Type**	**Mult.**	**Kind**	**Note**

▽

△

Class	**TransformationPropsToServiceInterfaceElementMapping**			
event	VariableDataPrototype	*	ref	This represents the reference to one or several events of one ServiceInterface. **Tags:**atp.Status=draft
field	Field	*	ref	This represents the reference to one or several fields of one ServiceInterface. **Tags:**atp.Status=draft
method	ClientServerOperation	*	ref	This represents the reference to one or several methods of one ServiceInterface. **Tags:**atp.Status=draft
tlvDataId Definition	TlvDataIdDefinitionSet	*	ref	This reference identifies the TlvDataIdDefinitions relevant for the enclosing TransformationPropsToServiceInterface Mapping. **Tags:**atp.Status=draft
transformation Props	TransformationProps	0..1	ref	This represents the reference to the applicable Serialization properties. **Tags:**atp.Status=draft
trigger	Trigger	*	ref	This represents the reference to one or several triggers of one ServiceInterface. **Tags:**atp.Status=draft

Table A.130: TransformationPropsToServiceInterfaceElementMapping

Enumeration	**TransportLayerProtocolEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInstanceDeployment
Note	This enumeration allows to choose a TCP/IP transport layer protocol. **Tags:**atp.Status=draft
Literal	*Description*
tcp	Transmission control protocol **Tags:** atp.EnumerationLiteralIndex=1 atp.Status=draft
udp	User datagram protocol **Tags:** atp.EnumerationLiteralIndex=0 atp.Status=draft

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*			
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. **Tags:**atp.Status=draft
Literal	*Description*
always	ServiceInterface element will trigger the transmission of the data. **Tags:** atp.EnumerationLiteralIndex=0 atp.Status=draft
never	ServiceInterface element will be buffered and will not trigger the transmission of the data. **Tags:** atp.EnumerationLiteralIndex=1 atp.Status=draft

Table A.133: UdpCollectionTriggerEnum

Class	**UserDefinedServiceInterfaceDeployment**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ServiceInstanceManifest::ServiceInterfaceDeployment			
Note	UserDefined configuration settings for a ServiceInterface. **Tags:** atp.Status=draft atp.recommendedPackage=ServiceInterfaceDeployments			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *ServiceInterfaceDeployment*, *UploadablePackageElement*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.134: UserDefinedServiceInterfaceDeployment

Class	**VariableDataPrototype**			
Package	M2::AUTOSARTemplates::SWComponentTemplate::Datatype::DataPrototypes			
Note	A VariableDataPrototype is used to contain values in an ECU application. This means that most likely a VariableDataPrototype allocates "static" memory on the ECU. In some cases optimization strategies might lead to a situation where the memory allocation can be avoided. In particular, the value of a VariableDataPrototype is likely to change as the ECU on which it is used executes.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *AutosarDataPrototype*, *DataPrototype*, *Identifiable*, *Multilanguage Referrable*, *Referrable*			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
initValue	ValueSpecification	0..1	aggr	Specifies initial value(s) of the VariableDataPrototype

Table A.135: VariableDataPrototype

B Platform Extension API (normative)

The focus of the APIs in this section are for OEM-specific platform extensions. The abstraction of the interfaces is lower which could lead to a higher machine dependency

B.1 Freshness Value Management(FVM) Library API

The following section provides the Freshness Value Management Library API as defined in fvm.h header file which is part of the `ara::com:secoc` namespace.

B.1.1 Library API Reference

[SWS_CM_11287]{DRAFT} ⌈

Kind:	class
Symbol:	FVM
Scope:	namespace ara::com::secoc
Syntax:	`class FVM {...};`
Header file:	#include "ara/com/secoc/fvm.h"
Description:	A freshness value management interface to be implmented by the OEM/stack vendor.
Notes:	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.

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11288]{DRAFT} ⌈

Kind:	function	
Symbol:	GetRxFreshness(std::uint16_t SecOCFreshnessValueID, const FVContainer &SecOCTruncatedFreshnessValue, std::uint16_t SecOCAuthVerifyAttempts)	
Scope:	class ara::com::secoc::FVM	
Syntax:	`ara::core::Result<FVContainer, SecOcFvmErrc> GetRxFreshness (std::uint16_t SecOCFreshnessValueID, const FVContainer &SecOCTruncatedFreshnessValue, std::uint16_t SecOCAuthVerifyAttempts) noexcept;`	
Parameters (in):	SecOCFreshnessValueID	the identifier of the freshness value.
	SecOCTruncatedFreshnessValue	the freshness value container with the values from the received Secured I-PDU/ message
	SecOCAuthVerifyAttempts	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, SecOc FvmErrc >	freshness value container that holds the freshness value to be used for the calculation of the the authenticator by the SecOC or recoverable error.
Exception Safety:	noexcept	
Header file:	#include "ara/com/secoc/fvm.h"	
Description:	This method is used by the SecOC to obtain the current freshness value.	
Notes:	synchronous, reentrant	

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11289]{DRAFT} ⌈

Kind:	function	
Symbol:	GetTxFreshness(std::uint16_t SecOCFreshnessValueID)	
Scope:	class ara::com::secoc::FVM	
Syntax:	`ara::core::Result<FVContainer, SecOcFvmErrc> GetTxFreshness (std::uint16_t SecOCFreshnessValueID) noexcept;`	
Parameters (in):	SecOCFreshnessValueID	the identifier of the freshness value.
Return value:	ara::core::Result< FVContainer, SecOc FvmErrc >	freshness value container that holds the freshness value to be used for the calculation of the authenticator by the SecOC or recoverable error.
Exception Safety:	noexcept	
Header file:	#include "ara/com/secoc/fvm.h"	
Description:	This method is used by the SecOC to obtain the current freshness value.	
Notes:	synchronous, reentrant	

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11290]{DRAFT} ⌈

Kind:	function	
Symbol:	Initialize()	
Scope:	class ara::com::secoc::FVM	
Syntax:	`ara::core::Result<void> Initialize () noexcept;`	
Return value:	ara::core::Result< void >	no return value in case of success, kFVInitialize Failed otherwise.
Exception Safety:	noexcept	
Header file:	#include "ara/com/secoc/fvm.h"	
Description:	This method initializes FVM plugin implementation.	
Notes:	synchronous, non-reentrant	

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11286]{DRAFT} ⌈

Kind:	struct
Symbol:	FVContainer
Scope:	namespace ara::com::secoc
Syntax:	`struct FVContainer {...};`
Header file:	#include "ara/com/secoc/fvm.h"
Description:	A freshness value container to hold the length of freshness value in bits and the freshness value itself as an ara::core::Vector .

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11344]{DRAFT} ⌈

Kind:	variable
Symbol:	length

▽

△

Scope:	struct ara::com::secoc::FVContainer
Type:	std::uint64_t
Syntax:	`std::uint64_t length;`
Header file:	#include "ara/com/secoc/fvm.h"
Description:	length in bits of the freshness value passed in FVContainer

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

[SWS_CM_11345]{DRAFT} ⌈

Kind:	variable
Symbol:	value
Scope:	struct ara::com::secoc::FVContainer
Type:	ara::core::Vector< std::uint8_t >
Syntax:	`ara::core::Vector<std::uint8_t> value;`
Header file:	#include "ara/com/secoc/fvm.h"
Description:	vector of bytes containing the freshness value
Notes:	depends if the container is used as an input or returning value by the method it will contain either the full freshness or truncated values

⌋(*RS_SEC_04001*, *RS_SEC_04002*, *RS_SEC_04003*, *RS_SEC_04004*)

B.1.2 Error Types

[SWS_CM_11340]{DRAFT} **Definition general ara::com::secoc errors** ⌈General ara::com::secoc errors shall be defined in the error domain `ara::com::secoc::SecOcFvmErrorDomain` in accordance with [16].⌋(*RS_AP_00130*)

[SWS_CM_11342]{DRAFT} ⌈

Kind:	enumeration	
Symbol:	SecOcFvmErrc	
Scope:	namespace ara::com::secoc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class SecOcFvmErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kFVNotAvailable= 1	Recoverable Error meaning the Freshness Value not available.
	kFVInitializeFailed= 2	Unrecoverable Error meaning the Freshness Value Manager could not be used.
Header file:	#include "ara/com/secoc/fvm_error_domain.h"	
Description:	The enumeration class defines the error codes for the SecOcFvmErrorDomain .	

⌋(*RS_AP_00130*)

[SWS_CM_11341]{DRAFT} **SecOcFvm errors domain** ⌈Error domain to describe ara::com errors related to the Freshness Value Management Library API `ara::com::secoc::SecOcFvmErrorDomain` shall be defined. It shall have the shortname SecOcFvm and the identifier 0x8000'0000'0000'1271.⌋*(RS_AP_00130)*

C History of Specification Items

C.1 Constraint and Specification Item History of this document according to AUTOSAR Release R17-10

C.1.1 Added Traceables 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

▽

△

Number	Heading
[SWS_CM_00428]	Element specification typed by String Data Type with `baseTypeSize` of 16
[SWS_CM_01031]	Service fields namespace
[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

▽

△

Number	Heading
[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 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

▽

△

Number	Heading
[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_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]	

▽

△

Number	Heading
[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_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 C.1: Added Traceables in 17-10

C.1.2 Changed Traceables 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

▽

△

Number	Heading
[SWS_CM_00206]	SOME/IP SubscribeEventgroupAck message
[SWS_CM_00300]	Event Cache Update Policy
[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 C.2: Changed Traceables in 17-10

C.1.3 Deleted Traceables in 17-10

Number	Heading
[SWS_CM_01003]	Inclusion protection

Table C.3: Deleted Traceables in 17-10

C.2 Constraint and Specification Item History of this document according to AUTOSAR Release R18-03

C.2.1 Added Traceables 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

▽

△

Number	Heading
[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.
[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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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`
[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 C.4: Added Traceables in 18-03

C.2.2 Changed Traceables 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

▽

△

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_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

▽

△

Number	Heading
[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
[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]	

▽

△

Number	Heading
[SWS_CM_10169]	
[SWS_CM_10172]	
[SWS_CM_10218]	
[SWS_CM_10219]	
[SWS_CM_10222]	
[SWS_CM_10234]	
[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]	

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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]	
[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 C.5: Changed Traceables in 18-03

C.2.3 Deleted Traceables 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

▽

△

Number	Heading
[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 C.6: Deleted Traceables in 18-03

C.3 Constraint and Specification Item History of this document according to AUTOSAR Release R18-10

C.3.1 Added Traceables 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
[SWS_CM_90210]	Using the DDS Security standard plug-ins in the Adaptive Platform

Table C.7: Added Traceables in 18-10

C.3.2 Changed Traceables 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
[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

▽

△

Number	Heading
[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 incomplete`Enumeration Data Type`s
[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
[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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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 `categoryVALUE`
[SWS_CM_11042]	DDS serialization of enumeration data types
[SWS_CM_11043]	DDS serialization of `StdCppImplementationDataType` of `categorySTRUCTURE`
[SWS_CM_11044]	DDS serialization of `StdCppImplementationDataType` of `categorySTRING` with string `shortName`
[SWS_CM_11046]	Encoding Format and Endianness of Strings in DDS
[SWS_CM_11047]	DDS serialization of `CppImplementationDataType` of `categoryVECTOR`
[SWS_CM_11048]	DDS serialization of `CppImplementationDataType` of `categoryARRAY`
[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
[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 C.8: Changed Traceables in 18-10

C.3.3 Deleted Traceables 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
[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

▽

△

Number	Heading
[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 C.9: Deleted Traceables in 18-10

C.4 Constraint and Specification Item History of this document according to AUTOSAR Release R19-03

C.4.1 Added Traceables in 19-03

none

C.4.2 Changed Traceables in 19-03

none

C.4.3 Deleted Traceables in 19-03

none

C.5 Constraint and Specification Item History of this document according to AUTOSAR Release R19-11

C.5.1 Added Traceables 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
[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]	

▽

△

Number	Heading
[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
[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]	

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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
[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 C.10: Added Traceables in R19-11

C.5.2 Changed Traceables 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
[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

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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
[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.

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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]	
[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

▽

△

Number	Heading
[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 C.11: Changed Traceables in R19-11

C.5.3 Deleted Traceables 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
[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]	

▽

△

Number	Heading
[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 C.12: Deleted Traceables in R19-11

C.6 Constraint and Specification Item History of this document according to AUTOSAR Release R20-11

C.6.1 Added Traceables 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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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]	
[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]	

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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]	
[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]	

▽

△

Number	Heading
[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]	
[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

▽

△

Number	Heading
[SWS_CM_99027]	Cg errors domain

Table C.13: Added Traceables in R20-11

C.6.2 Changed Traceables 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

▽

△

Number	Heading
[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
[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]	

▽

△

Number	Heading
[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]	
[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

▽

△

Number	Heading
[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
[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 C.14: Changed Traceables in R20-11

C.6.3 Deleted Traceables 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 C.15: Deleted Traceables in R20-11

C.7 Constraint and Specification Item History of this document according to AUTOSAR Release R21-11

C.7.1 Added Traceables 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

▽

△

Number	Heading
[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
[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)

▽

△

Number	Heading
[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
[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 C.16: Added Traceables in R21-11

C.7.2 Changed Traceables 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
[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

▽

△

Number	Heading
[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
[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

▽

△

Number	Heading
[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
[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]	

▽

△

Number	Heading
[SWS_CM_11312]	
[SWS_CM_11318]	
[SWS_CM_11319]	
[SWS_CM_11320]	
[SWS_CM_11322]	
[SWS_CM_11323]	
[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

▽

△

Number	Heading
[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]	
[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 C.17: Changed Traceables in R21-11

C.7.3 Deleted Traceables 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

▽

△

Number	Heading
[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 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

▽

△

Number	Heading
[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_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 C.18: Deleted Traceables in R21-11